1 基本概念

进入Java的世界

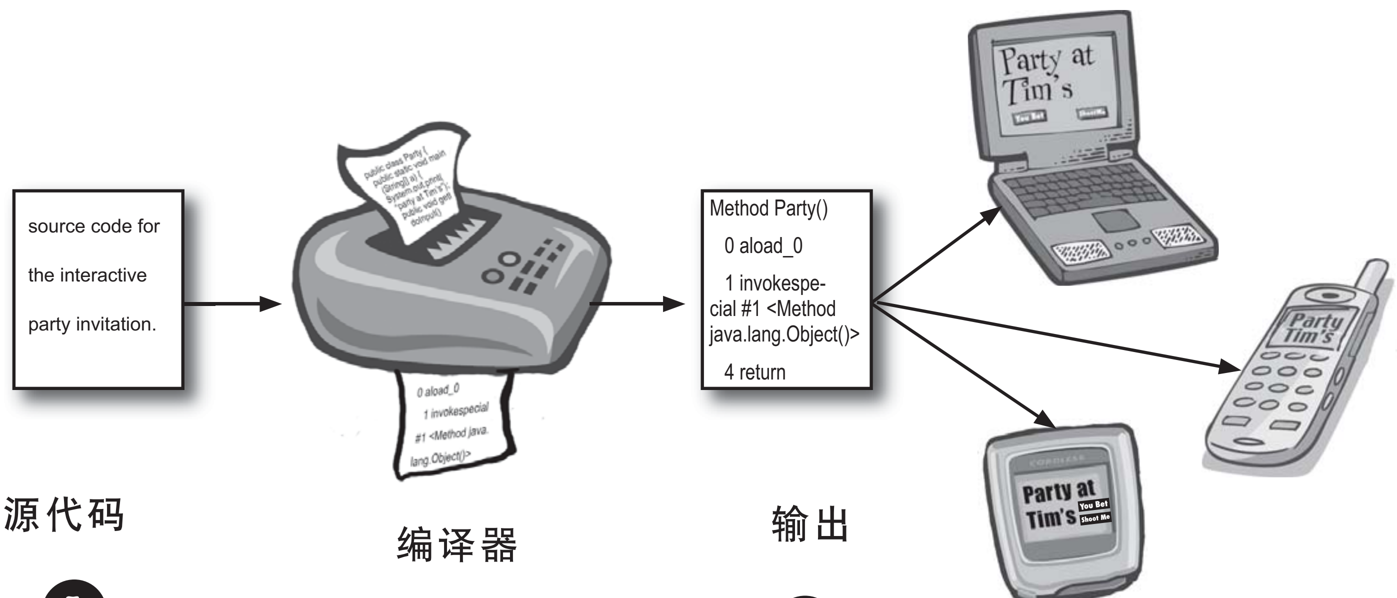


Java将带你进入新领域。它从一开始就以友好的语法、面向对象、内存管理和最棒的跨平台可移植性来吸引程序员。写一次就可以在所有地方执行（write-once/run-anywhere）的特性简直是迷死人了。许多人在投入后才发现有bug要除、功能限制很大、最要命的是运行起来超慢！不过这都是很久以前的事情了。如果你现在才刚开始接触Java，那你还真幸运。现在的Java可是又快又有威力。



Java的工作方式

它的目标是要让你写出一个应用程序（在此例中是一个交互式派对邀请函系统）且能够在你的朋友所拥有的任何设备上执行。



源代码

❶ 编写源代码文件。

编译器

❷ 用编译器运行源代码。编译器会检查错误，如果有错就要改正才能产生正确的输出。

输出

❸ 编译器会产出字节码。任何支持Java的装置都能够把它转译成可执行的内容。编译后的字节码与平台无关。

Java虚拟机（JVM）

❹ 你的朋友不会买一台真正的Java机器，但是他们都会有Java虚拟机（通过软件实现）。此虚拟机可以读取与执行字节码。

你要做的事

你会编写源代码文件，用javac编译程序把文件进行编译，然后在某个Java虚拟机上执行编译过的字节码。

```
import java.awt.*;
import java.awt.event.*;
class Party {
  public void buildInvite() {
    Frame f = new Frame();
    Label l = new Label("Party at Tim's");
    Button b = new Button("You bet");
    Button c = new Button("Shoot me");
    Panel p = new Panel();
    p.add(l);
  }  // more code here...
}
```

源代码

1

编写源代码。

存为Party.java



编译器

2

执行javac程序来编译Party.java。如果程序没有错误的话，会产生Party.class这个文件。

编译器编译出来的这个文件是由字节码所组成的。

```
Method Party()
  0 aload_0
  1 invokespecial #1 <Method java.
lang.Object()>
  4 return
Method void buildInvite()
  0 new #2 <Class java.awt.Frame>
  3 dup
  4 invokespecial #3 <Method java.
awt.Frame()>
```

输出

3

编译成代码：Party.class

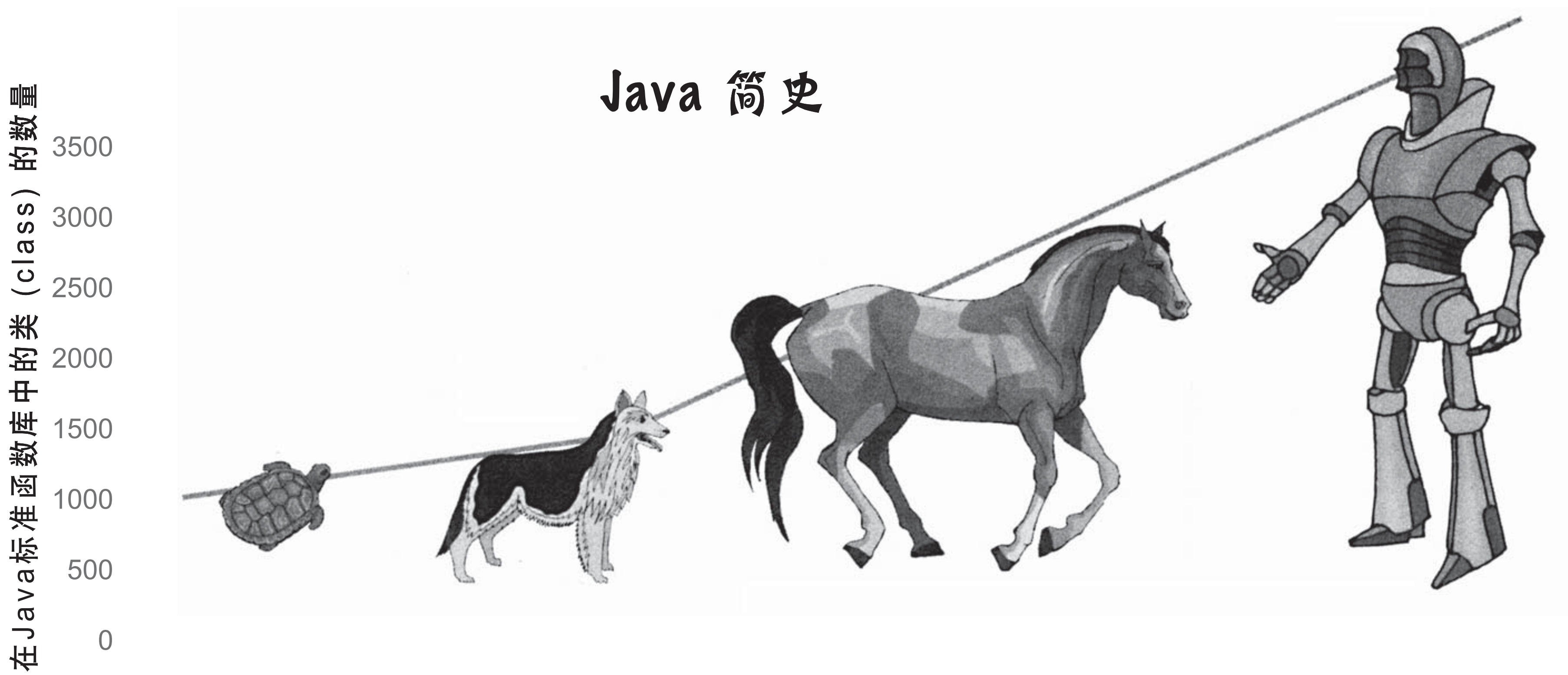


Java虚拟机

4

启动Java虚拟机（JVM）来运行Party.class文件。JVM会将字节码转换成平台能够理解的形式来运行。

（注意：这一页不是练习题，等一下就会让你编写程序，但现在不用，只是要让你知道来龙去脉。）



Java 1.02

250 个类

龟速

有可爱的logo和名称，非常有趣，但是bug很多，其中applet是重点。

Java 1.1

500 个类

狗速

功能更强、更好用。开始受到欢迎。比较适于开发图形界面。

Java 2 (版本 1.2～1.4)

2300 个类

马速

有时可以达到平台原始(native)速度。可以用来书写正规的企业级应用程序或移动应用程序。有3种版本：Micro Edition (J2ME)、Standard Edition(J2SE)以及 Enterprise Edition (J2EE)。

Java 5.0 (版本1.5及以上)

3500 个类

机器人等级的威力，更容易开发

除了新增数以千计的类之外，Java 5.0（又称为Tiger）还对语言本身作了许多重大的改变，使得它在理论上更容易使用，并含有其他语言中很受欢迎的功能。

Sharpen your pencil

看，写Java程序就是这么简单！

```
int size = 27;
String name = “Fido”;
Dog myDog = new Dog(name, size);
x = size - 5;
if (x < 15) myDog.bark(8);

while (x > 3) {
   myDog.play();
}

int[] numList = {2,4,6,8};
System.out.print(“Hello”);
System.out.print(“Dog: “ + name);
String num = “8”;
int z = Integer.parseInt(num);

try {
   readTheFile(“myFile.txt”);
}
catch(FileNotFoundException ex) {
   System.out.print(“File not found.”);
}
```

猜猜看每一行程序在做什么？(答案在下一页)

声明一个 integer类型，名称为size的变量并赋值27

问： 我曾经看过Java 2和Java 5.0，是否有Java 3和4呢？为什么2没有小数点？

答： 搞销售的乐趣就在于此，虽然Java的实际版本是从1.1推进到1.2，但因变化幅度大，所以销售部门决定用全新的命名，于是命名为Java 2。Java 3或4从来没有出现过。到了1.5版时，销售部门又认为进步幅度太大（大部分的程序员都同意这么说）而需要全新的名称，所以又冒出一个 Java 5.0，用5对应到1.5。

最原始的Java版本是Java 1.02（第一次出版的版本），1.02到1.1版都算是“Java”。1.2、1.3和1.4版都叫做“Java 2”。从1.5版开始叫做“Java 5.0”。你偶尔也会看到“Java 5”或“Tiger（开发代号）”的用法，我不知道下一个版本该叫什么。

Sharpen your pencil 解答

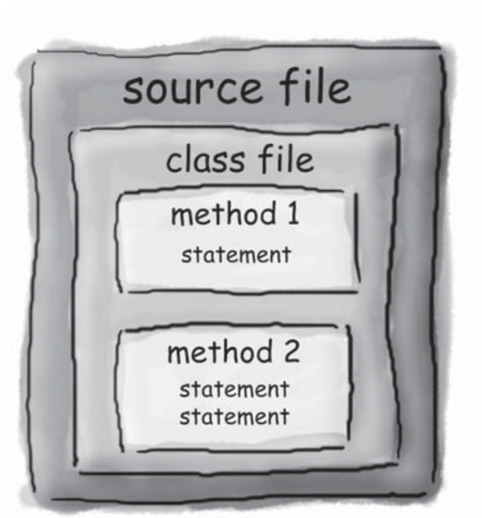
看，写 Java 程序就是这么简单！

看不懂也无所谓！

后面会有很详细的解说（约40页）。如果你学过类似的语言就会发现这些东西很简单。如果没有的话，接下来就会让你知道……

`int size = 27;`	声明一个integer类型、名称为size的变量并赋初始值为27
`String name = “Fido”;`	声明名称为name的字符串，值为Fido
`Dog myDog = new Dog(name, size);`	以name与size声明一个名称为myDog的Dog变量
`x = size - 5;`	把size值减5的结果赋给变量x
`if (x < 15) myDog.bark(8);`	如果x的值小于15，让狗吠8次
`while (x > 3) {`	当x的值大于3就持续执行循环
`   myDog.play();`	让狗做些执行 play 动作（不管那对狗有什么意义）
`}`	{ } 里面就是循环的内容
`int[] numList = {2,4,6,8};`	声明有4个元素的整型数组
`System.out.print(“Hello”);`	把“Hello”输出到屏幕上
`System.out.print(“Dog: “ + name);`	把“Dog:”输出到屏幕上，后面跟着狗的名字（Fido）
`String num = “8”;`	声明字符串变量num并赋值为 8
`int z = Integer.parseInt(num);`	将字符串“8”转换成整数数字 8
`try {`	试列出可能会出异常状况的指令
`   readTheFile(“myFile.txt”);`	读取myfFile.txt这个文件
`}`	{ }中的指令被视为一体
`catch(FileNotFoundException ex) {`	捕获万一发生的找不到文件异常状况
`   System.out.print(“File not found.”);`	万一找不到文件就说找不到
`}`	{ } 里面是异常的处理程序

Java的程序结构



类存于源文件里面

方法 存于 类 中

语句 (statement) 存于方法中

什么是源文件?

源文件(扩展名为 .java)带有类的定义。类用来表示程序的一个组件，小程序或许只会有一个类。类的内容必须包在花括号里面。

```
public class Dog {

}
```

类

什么是类?

类中带有一个或多个方法。在Dog这个类中，bark方法带有如何“汪汪”的指令。方法必须在类的内部声明。

```
public class Dog {
   void bark() {

   }
}
```

方法

什么是方法?

在方法的花括号中编写方法应该执行的指令。方法代码是由一组语句所组成，你可以把方法想象成是一个函数或过程。

```
public class Dog {
   void bark() {
     statement1;
     statement2;
   }
}
```

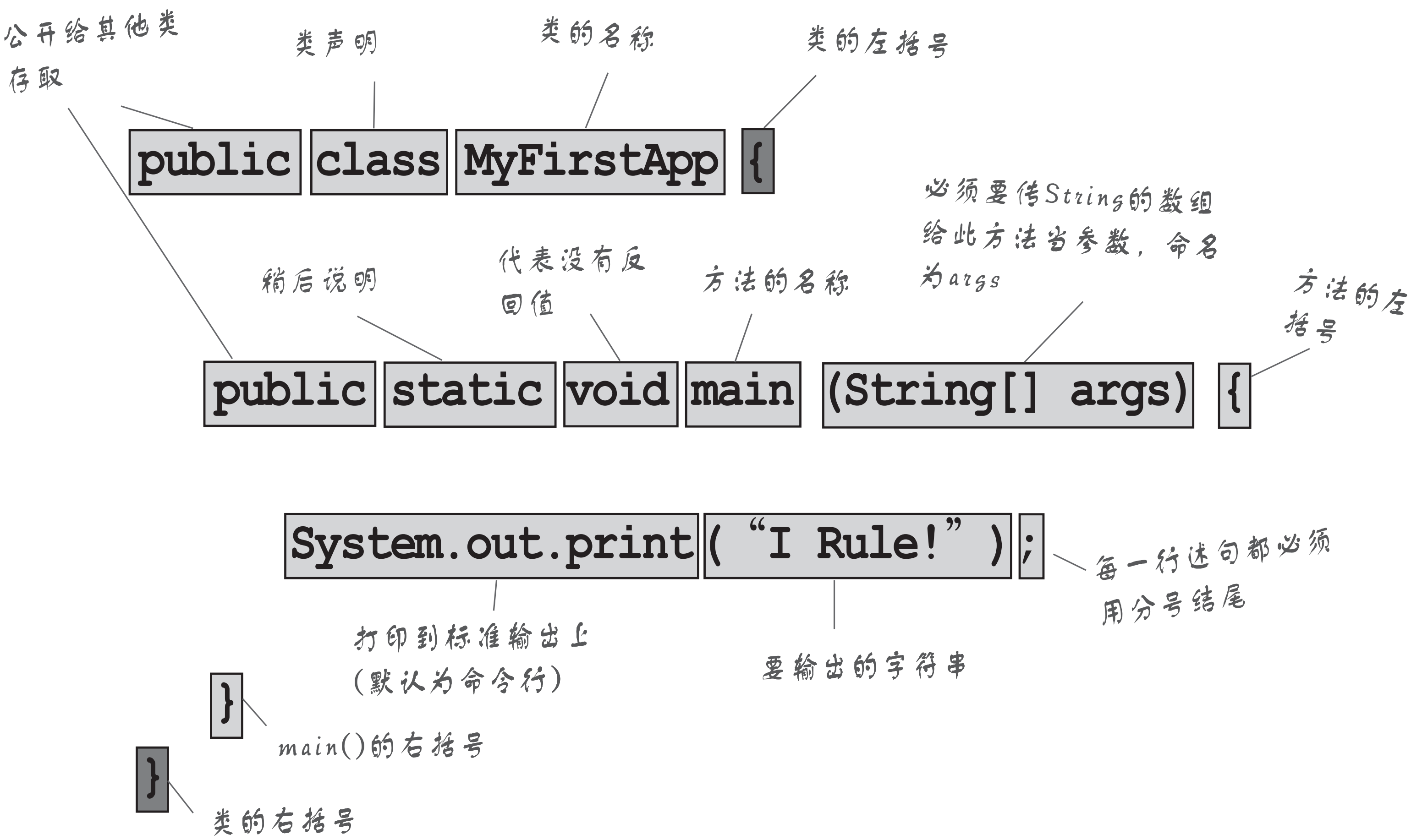
语句

剖析类

当Java虚拟机启动执行时，它会寻找你在命令列所指定的类。然后它会锁定像下面这样一个特定的方法：

```
public static void main (String[] args) {
      // 程序代码写在这里
  }
```

接着Java虚拟机就会执行main方法在花括号间的函数所有指令。每个Java程序最少都会有一个类以及一个main()。每个应用程序只有一个main()函数。



现在还不需要把这些东西背下来，这一章只是热身活动。

编写带有main()的类

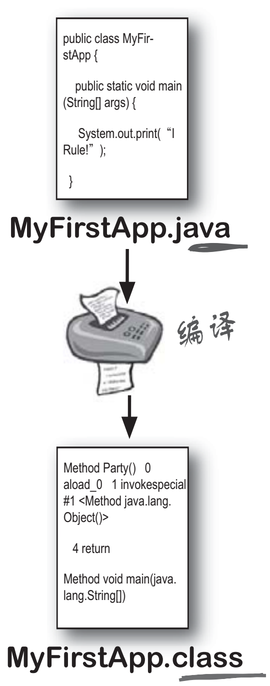
在Java中的所有东西都会属于某个类。你会建立源文件（扩展名为.java），然后将它编译成新的类文件（扩展名为 .class）。真正被执行的是类。

要执行程序就代表要命令Java虚拟机（JVM）去“加载Hello这个类，开始执行它的main()，然后一直运行到main的所有程序代码结束为止”。

在第2章中我们会更深入地探讨类的细节，但是现在你只要注意到如何编写与执行Java程序就行了。而这些都与main()有关。

main()就是程序的起点。

不管你的程序有多大（也可以说不管有多少个类），一定都会有一个main()来作为程序的起点。




```
public class MyFirstApp {

   public static void main (String[] args) {
     System.out.println("I Rule!");
     System.out.println("The World");
   }

}
```

1 保存

`MyFirstApp.java`

2 编译

`javac MyFirstApp.java`

3 运行

```
File Edit Window Help Scream
%java MyFirstApp
I Rule!
The World
```

你能在main()中做什么？

你一旦进入main()或其他的方法中，就可以找到乐子。你可以作出任何正常的行为来让编译器（compiler）忙。

你的程序代码可以让Java虚拟机去：



❶ 做某件事

声明、设定、调用方法等普通语句。

```
int x = 3;
String name = “Dirk”;
x = x * 17;
System.out.print(“x is ” + x);
double d = Math.random();
// 这是注释行
```

❷ 反复做某件事

for 与 while 的循环（loop）

```
while (x > 12) {
   x = x -1;
}

for (int x = 0; x < 10; x = x + 1) {
   System.out.print(“x is now ” + x);
}
```

❸ 在适当条件下做某件事

if/else 的条件分支测试

```
if (x == 10) {
   System.out.print(“x must be 10”);
} else {
   System.out.print(“x isn’t 10”);
}
if ((x < 3) & (name.equals(“Dirk”))) {
   System.out.println(“Gently”);
}
System.out.print(“this line runs no matter what”);
```

语法乐趣多

★ 语句是以分号结束。

```
x = x + 1;
```

★ 以两条斜线开始的行是注释。

```
x = 22;
//  我是注释
```

★ 空格符通常无关紧要。

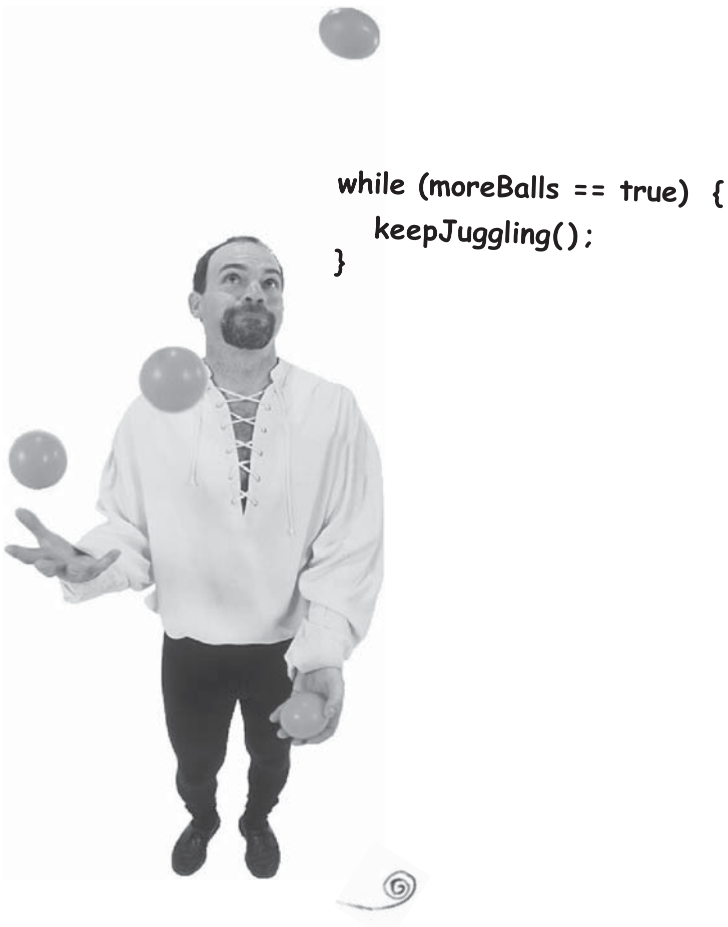
```
x      =      3   ;
```

★ 用名称与类型（type）来声明变量（第3章会讨论类型）。

```
int weight;
// 类型： int，名称：weight
```

★ 类型与方法都必须定义在花括号中。

```
public void go() {
   // 程序代码放在这里
}
```



重复再重复，循环再循环……

Java有3种循环结构：while循环、do-while循环和for循环。稍后你会看到详细的说明，但是先让我们来看一下while循环是怎么工作的。

这个语法简单到你可能已经睡着了。只要while条件为 true，循环块中的程序代码就会一直重复执行。程序代码块是由一对花括号所规范的，所以要重复的区段必须摆在括号中。

循环的关键在于条件测试（conditional test）。在Java中，条件测试的结果是boolean值——不是true就是false。

例如说“如果含笑半步癫比一日丧命散还好吃我就嗑给你看”是个有效的boolean测试，但“如果香蕉我就说芭蕉”并不是个条件测试。条件判断式必须要能够求出真伪值。

简单的boolean测试

你可以用比较运算符（comparison operator）来执行简单的boolean值测试：

< (小于)

> (大于)

== (等于)

注意：赋值运算符（一个等号）与等号运算符（两个等号）并不一样。许多程序员都会不小心犯这个错，但我相信你不会。

```
int x = 4;  //给x赋值为4
while (x > 3) {
  // 循环会运行是因为
  // x 大于 3
  x = x - 1; //避免无限循环
}
int z = 27; //
while (z == 17) {
  // 循环不会运行
  // 因为z不等于17
}
```

there are no Dumb Questions

问：为何所有的东西都得包含在类中？

答： 因为Java是面向对象的语言，它不像是以前的程序语言那样。第2章会说明类是对象的蓝图，而Java中的绝大多数东西都是对象。

问：每个类都需要加上一个main()吗？

答： 程序不是这样写的。一位大名鼎鼎的程序大师已经解释过一个程序只要一个main来作为运行。

问：其他程序语言可以直接用整数类型测试，我也可以像下面这么做吗：

```
int x = 1;
while (x){ }
```

答： 不行，Java中的integer与boolean两种类型并不相容。你只能用下面这样的boolean变量来测试：

```
boolean isHot = true;
while(isHot) { }
```

while循环的范例：

```
public class Loopy {
   public static void main (String[] args) {
     int x = 1;
     System.out.println("Before the Loop");
     while (x < 4) {
       System.out.println("In the loop");
       System.out.println("Value of x is " + x);
         x = x + 1;
     }
     System.out.println("This is after the loop");
   }
}
```

这是输出

```
% java Loopy
Before the Loop
In the loop
Value of x is 1
In the loop
Value of x is 2
In the loop
Value of x is 3
This is after the loop
```

要点

- 语句以分号结束。
- 程序块以{ }划出范围。
- 用名称与类型声明变量。
- 等号是赋值运算符。
- 两个等号用来当等式等号运算符。
- 只要条件测试结果为真，while循环就会一直执行块内的程序。
- 把boolean测试放在括号中：

  `while (x == 4) { }`

条件分支

在Java中if与while循环都是boolean测试，但语义从“只要不下雨就持续……”改成“如果不下雨就……”。

```
class IfTest {
  public static void main (String[] args) {
    int x = 3;
    if (x == 3) {
      System.out.println(“x must be 3”);
    }
    System.out.println(“This runs no matter what”);
  }
}
```

```
% java IfTest
x must be 3
This runs no matter what
```

这是输出

上面的程序只会在x等于3这个条件为真的时候才会列出“x must be 3”。而第二行无论如何都会列出。

我们可以将程序加上else条件，因此可以指定“如果下雨就撑雨伞，不然的话就戴墨镜”。

```
class IfTest2 {
  public static void void  main (String[] args) {
    int x = 2;
    if (x == 3) {
      System.out.println(“x must be 3”);
    } else {
       System.out.println(“x is NOT 3”);
    }
    System.out.println(“This runs no matter what”);
  }
}
```

```
% java IfTest2
x is NOT 3
This runs no matter what
```

修改后的输出

System.out.print 与 System.out.println

如果仔细观察的话，你会注意到我们将print改成了println。

差别在哪里？

println会在最后面插入换行，若你想要让后续的输出以新的一行开始，可以使用println，若是使用print则后续的输出还是会在同一行。

Sharpen your pencil

若程序输出如下：

```
% java DooBee
DooBeeDooBeeDo
```

请填入程序空格部分：

```
public class DooBee {
  public static void main (String[] args) {
    int x = 1;
    while (x < _____ ) {
      System.out._________(“Doo”);
      System.out._________(“Bee”);
      x = x + 1;
    }
    if (x == ______ ) {
      System.out.print(“Do”);
    }
  }
}
```

设计真正的应用程序



让我们将你已经学习到的Java技能发挥到真正有用的程序上。我们需要一个带有main()的类、一个int与String变量、一个while循环和一个if测试。稍加整理你就能立即创建出有用的程序。但在你偷看示例程序代码之前，先自己想想看你会怎样写一个程序从99数到0。下面是数啤酒瓶童谣的程序：

```
public class BeerSong {
   public static void main (String[] args) {
     int beerNum = 99;
     String word = “bottles”;

     while (beerNum > 0) {

        if (beerNum == 1) {
          word = “bottle”; // 单数的瓶子
        }

        System.out.println(beerNum + “ ” + word + “ of beer on the wall”);
        System.out.println(beerNum + “ ” + word + “ of beer.”);
        System.out.println(“Take one down.”);
        System.out.println(“Pass it around.”);
        beerNum = beerNum - 1;

        if (beerNum > 0) {
           System.out.println(beerNum + “ ” + word + “ of beer on the wall”);
        } else {
           System.out.println(“No more bottles of beer on the wall”);
        }//else结束
     } //while循环结束
  } //main方法结束
} //class结束
```

虽然程序可以编译与运行，但它的输出有点不美观，你可以试试看改漂亮一点。

阿强的周一早晨

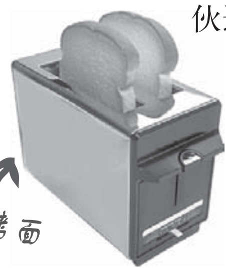
平时阿强的闹钟设定在8:30，但在狂野的周末他会把早上闹钟的声音按掉。此时有Java功能的设备会开始起作用。



首先，闹钟会送出一个信息给咖啡机说："嗨，这位老兄又赖床了，12分钟后再煮咖啡"。

咖啡机会送出一个信号给烤面包机说："先不要烤，那家伙还在睡"。





之后闹钟还会送出信号给阿强的手机："9:00时打给阿强告诉他我们晚一点才会准备好"。

最后，闹钟会送个信号给大头（大头是只狗）的无线颈圈表示去捡报纸回来，但是别想出去散步。



几分钟后闹钟又响了，阿强还是把它按掉，而设备们又开始交谈。第三次阿强还是想要把闹钟按掉，但是闹钟先发制人的送出一个信息给大头的颈圈表示要它跳上床叫人。

终于把阿强叫醒了，他心想幸好Java设备有发挥作用，算是没有白跑一趟光华商场。



面包已经烤好了。

咖啡热腾腾的。

报纸也拿进来了。

又是个美好的早晨。你可以用Java、网络以及Jini技术来装置一栋Java-Enabled的房子。但是要小心所谓的"即插即用"技术（事实上即插即用代表着插入之后随既使用技术支持客服专线）或是"可移植性"平台。阿强的老姐也使用了这些设备，但是效果很差，甚至还具有危险。他的狗在生前也是这么认为……



这个故事有可能是真的吗？是也不是。虽然有各种版本的Java用在PDA、移动电话、智能卡等装置上，但你目前还不太可能找到有Java的烤面包机与狗项圈。然而还是可以通过有Java的计算机等接口来控制其他设备。这就是称为Jini surrogate的架构。你还是可以这样设计梦想豪宅。



好，数啤酒瓶不算是真正的应用程序，如果要给老板看很棒的程序，那就把专家术语学习机程序秀出来。

注意：当你自己输入这个程序时，不要在字符串中间换行（也就是“”符号之间），不然的话程序无法通过编译程序。

```
public class PhraseOMatic {
   public static void main (String[] args) {

1     // 你可以随意的加上其他术语
      String[] wordListOne = {"24/7","multi-
Tier","30,000 foot","B-to-B","win-win","front-
end", "web-based","pervasive", "smart", "six-
sigma","critical-path", "dynamic"};

   String[] wordListTwo = {"empowered", "sticky",
"value-added", "oriented", "centric", "distributed",
"clustered", "branded","outside-the-box", "positioned",
"networked", "focused", "leveraged", "aligned",
"targeted", "shared", "cooperative", "accelerated"};

   String[] wordListThree = {"process", "tipping-
point", "solution", "architecture", "core competency",
"strategy", "mindshare", "portal", "space", "vision",
"paradigm", "mission"};

    // 计算每一组有多少个名词术语
2  int oneLength = wordListOne.length;
   int twoLength = wordListTwo.length;
   int threeLength = wordListThree.length;

   // 产生随机数字
3  int rand1 = (int) (Math.random() * oneLength);
   int rand2 = (int) (Math.random() * twoLength);
   int rand3 = (int) (Math.random() * threeLength);

   // 组合出专家术语
4  String phrase = wordListOne[rand1] + " " +
wordListTwo[rand2] + " " + wordListThree[rand3];

   // 输出
5  System.out.println("What we need is a " + phrase);
   }
}
```

专家术语学习机

它是如何工作的

简单地讲，它是由3组单字随机挑出来排列组合输出。暂时不用担心你看不懂某一行程序在做什么，毕竟我们才刚开始而已。

1. 第一个步骤是创建出3个String的数组，也就是保存术语的容器。数组的声明和创建是很简单的，下面是一个例子：

```
String[] pets = {"Fido", "Zeus", "Bin"};
```

每个元素放在引号中并彼此间以逗号分开。

2. 为了要在每个数组中能够随机地挑出一个单字，我们得知道每个数组的大小。如果某个数组有14个单字，则我们需要介于0～13的随机数（Java的数组是零基的，第一个元素的索引是0，第二个是1，在有14个元素的数组中最后一个的索引是13）。你可以向数组查询它的长度：

```
int x = pets.length;
```

执行以后 x 的值为 3。

3. 我们需要3个随机数。Java本身有一组立即可用的数学方法（可以把它们当作函数）。random()这个方法会返回介于0与1之间的值，所以我们需要将此值乘以数组的元素数量（数组的大小），然后取整数值（第4章会讨论）。如果要对任何浮点数取整数值也是用这样的方法转换数据类型：

```
int x = (int) 24.6;
```

4. 现在我们可以创建专用的术语。选出3个字然后使用“+”这个运算符将字符串对象连接在一起。使用索引数字可以将数组中的元素提取出来：

```
String s = pets[0]; // "Fido"
s = s + " " + "is a dog"; // "Fido is a dog"
```

5. 最后，我们将结果输出到命令列上，你就可以引用这些术语说出听起来很厉害但是完全没有意义的句子。

我们可以输出下面这些言不及义的术语……

pervasive targeted process

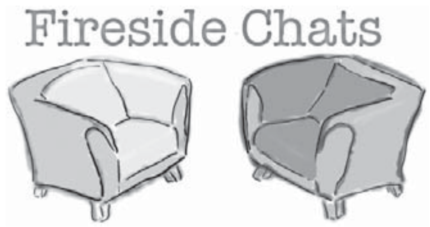
dynamic outside-the-box tipping-point

smart distributed core competency

24/7 empowered mindshare

30,000 foot win-win vision

six-sigma net-worked portal



今晚的话题：

编译器与JVM争辩谁比较重要？

Java 虚拟机

什么？你开玩笑吧？这位大婶，我可是 Java 啊。只有我才能让程序运行起来。你只是产生文件而已。做个文件有什么了不起的，没有我，文件没有用！

编译器

请你放尊重点，不然我要叫了。

Java 虚拟机

还有，你得理不饶人，每天老是警告人，小小一点语法错误也不放水……

编译器

对不起，没有我你能运行什么？Java会设计成这样是有原因的。如果Java只是个直译语言，要一边运行一边解译纯文字的程序，我就不相信你能够运行多快！

Java 虚拟机

抗议啦，我又没有说你一点用处都没有。但说真的，我根本搞不懂你在做什么。程序员可以直接编写二进制代码给我运行，那你就失业啦，哇哈哈哈……哇哈哈哈……咳……哇哈哈……

编译器

我实在懒得理你。没错，虽然说只要是合格的二进制代码就可以运行，不一定是要编译器编译出来的，但实际上不会有人傻成这样的。让程序员直接写出二进制代码就好像要组装计算机的人自己得作出CPU一样。还有，你可不可以不要笑得哪么难听？

Java 虚拟机

先不管它的笑声问题。你还是没有回答我你到底有什么用处？

Java 虚拟机 / **编译器**

编译器：还记得Java是个强类型的语言吗，这代表我不能容许变量保存类型的数据。这是很关键的类型安全性功能，我能够让大部分的错误在到你那边之前就被抓到。还有……

Java 虚拟机：又不是全部抓光光！我还是会因为遇到将错误类型的数据塞进数组中而不得不抛出异常，并且……

编译器：没礼貌，别打断我说话……，是有些数据类型的错误会在运行时发生，但这也是为了要容许动态绑定这样的功能。Java可以在执行期引用连程序员也没有预期会碰到的类型，所以我得留一些运用性。我的工作就是要确保铁定不能跑的东西不会过关。通常我会抓得到错误，例如说把文字字符串除以某个数字这种问题就会被我发现。

Java 虚拟机：OK，当然。但是存取权限的安全问题呢？还不是靠我把关，而你只不过是作些标点符号的检查罢了。还真谢谢你把这些问题留给我呢。

编译器：对不起，大家都知道我才是安全的第一线。我刚刚说的数据类型错误如果没有处理好可是一个漏洞呢。像是违反调用private方法的程序等也是由我检查的。我能够防止人们动到不可以碰的程序代码与其他类的重要数据。如果要把我的功能说完可能要说到天亮。

Java 虚拟机：随你怎么说。我也得做相同的事情，确保不会有人在执行前修改二进制代码。

编译器：是啦，如果没有我挡住上述的问题，你老早就挂掉了。没时间了，下回再说吧。

Java 虚拟机：OK，等一下要不要去吃宵夜？



排排看



小朋友，右边是原本写好的程序，但是被饼干大怪兽给弄乱了，英勇的你是不是可以把程序排回原状来产生像下面这样的输出呢？注意到有些括号被吃掉了，请爸爸妈妈一起帮忙补起来！

```
if (x == 1) {
  System.out.print("d");
  x = x - 1;
}
```

```
if (x == 2) {
  System.out.print("b c");
}
```

```
class Shuffle1 {
  public static void main(String [] args) {
```

```
if (x > 2) {
  System.out.print("a");
}
```

```
int x = 3;
```

```
x = x - 1;
System.out.print("-");
```

```
while (x > 0) {
```

输出：

```
File  Edit  Window  Help  Sleep
% java Shuffle1
a-b c-d
```



练习

我是编译器



这一页的Java程序代码代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪个程序可以编译过关。如果有问题，哪里需要修改？

A

```
class Exercise1b {
  public static void main(String [] args) {
    int x = 1;
    while ( x < 10 ) {
      if ( x > 3) {
        System.out.println("big x");
      }
    }
  }
}
```

B

```
public static void main(String [] args) {
    int x = 5;
    while ( x > 1 ) {
      x = x - 1;
      if ( x < 3) {
        System.out.println("small x");
      }
    }
}
```

C

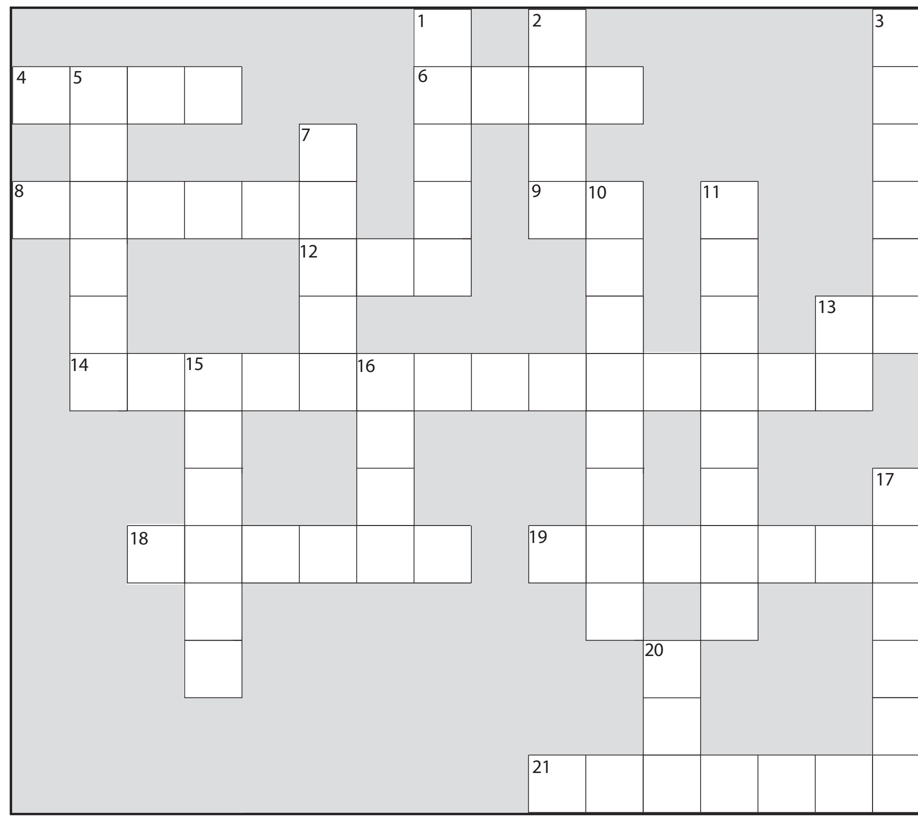
```
class Exercise1b {
    int x = 5;
    while ( x > 1 ) {
      x = x - 1;
      if ( x < 3) {
        System.out.println("small x");
      }
    }
}
```



JavaCross 7.0

找点事情让你的大脑动一动。

这是个字谜，几乎所有的单字都在这一章提过。为了提高你的注意力，我们偷偷地放了一些非Java的高科技词汇。



横排提示

4. Command-line invoker
6. Back again?
8. Can't go both ways
9. Acronym for your laptop's power
12. number variable type
13. Acronym for a chip
14. Say something
18. Quite a crew of characters
19. Announce a new class or method
21. What's a prompt good for?

竖排提示

1. Not an integer (or _____ your boat)
2. Come back empty-handed
3. Open house
5. 'Things' holders
7. Until attitudes improve
10. Source code consumer
11. Can't pin it down
13. Dept. of LAN jockeys
15. Shocking modifier
16. Just gotta have one
17. How to get things done
20. Bytecode consumer

（字谜的问题部分保留原汁原味的英文，请自己动手查字典！）



连连看

别翻脸，我不会再提饼干大怪兽了，请你也不要找爸妈帮你作答好吗？下面有段程序代码不见了，你的工作是要把左边的程序代码填入消失的段落以连接到右边的输出。不一定每个输出都会对应到某个程序代码段，有的输出可能会用到两次以上。

```
class Test {
  public static void main(String [] args) {
    int x = 0;
    int y = 0;
    while ( x < 5 ) {
      [                              ]
      System.out.print(x + " " + y + " ");
      x = x + 1;
    }
  }
}
```

消失的程序代码段落

将程序代码划条线连到正确的输出结果

程序代码：

```
y = x - y;
```

```
y = y + x;
```

```
y = y + 2;
if( y > 4 ) {
  y = y - 1;
}
```

```
x = x + 1;
y = y + x;
```

```
if ( y < 5 ) {
  x = x + 1;
  if ( y < 3 ) {
    x = x - 1;
  }
}
y = y + 2;
```

输出结果：

```
22 46
```

```
11 34 59
```

```
02 14 26 38
```

```
02 14 36 48
```

```
00 11 21 32 42
```

```
11 21 32 42 53
```

```
00 11 23 36 410
```

```
02 14 25 36 47
```



泳池迷宫



你的任务是要从游泳池挑出程序片段并将它填入右边的空格中。同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。别被骗了，这个问题比看起来的样子要困难的多。

输出：

```
File  Edit  Window  Help  Cheat
%java PoolPuzzleOne
a noise
annoys
an oyster
```

```
class PoolPuzzleOne {
  public static void main(String [] args) {
    int x = 0;

    while ( __________ ) {

      ______________________________
      if ( x < 1 ) {
        ____________________________
      }
      ______________________________

      if ( __________ ) {

        ____________________________

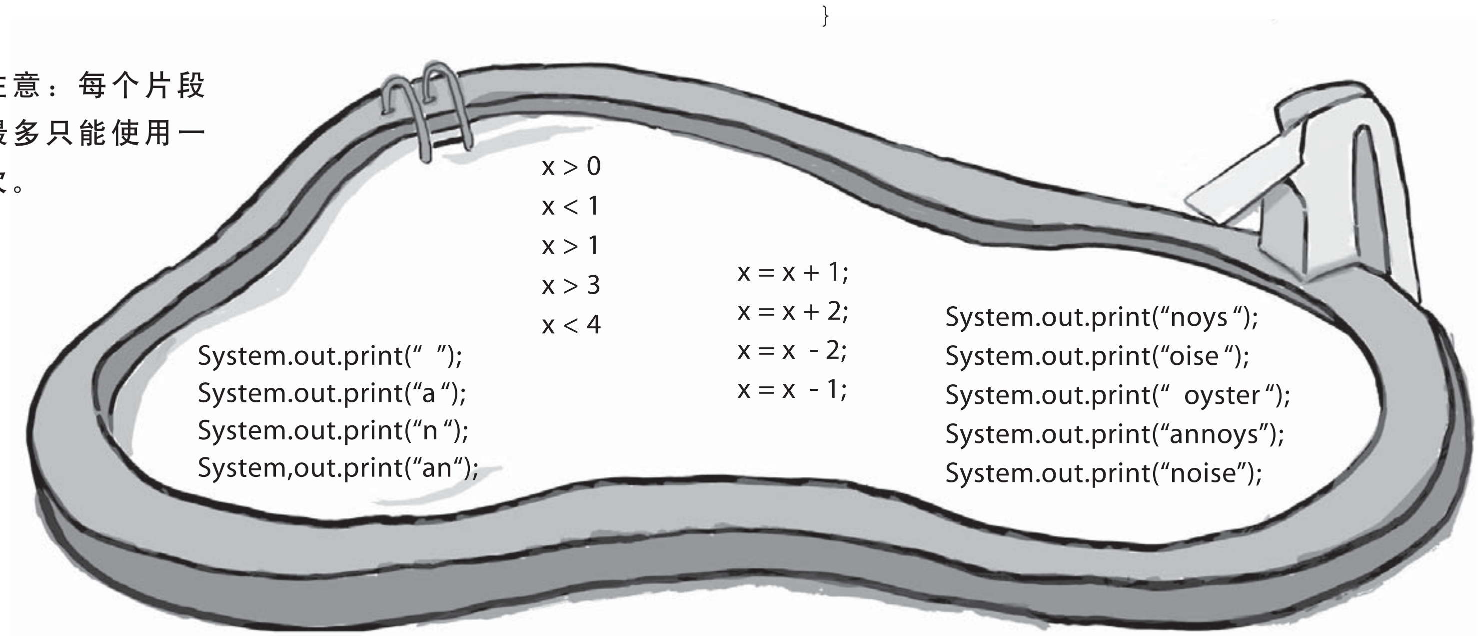
        ___________
      }
      if ( x == 1 ) {

        ____________________________
      }
      if ( ___________ ) {

        ____________________________
      }
      System.out.println("");

      ____________
    }
  }
}
```

注意：每个片段最多只能使用一次。





解答

排排看

```
class Shuffle1 {
  public static void main(String [] args) {

    int x = 3;
    while (x > 0) {

      if (x > 2) {
        System.out.print("a");
      }

      x = x - 1;
      System.out.print("-");

      if (x == 2) {
        System.out.print("b c");
      }

      if (x == 1) {
        System.out.print("d");
        x = x - 1;
      }
    }
  }
}
```

```
File Edit Window Help Poet
% java Shuffle1
a-b c-d
```

A

```
class Exercise1b {
  public static void main(String [] args) {
    int x = 1;
    while ( x < 10 ) {
      x = x + 1;
      if ( x > 3) {
        System.out.println("big x");
      }
    }
  }
}
```

这个程序可以编译与执行，但如果没有加入特别标记出来的这一行，它会无止境地执行下去！

B

```
class Foo {
  public static void main(String [] args) {
    int x = 5;
    while ( x > 1 ) {
      x = x - 1;
      if ( x < 3) {
        System.out.println("small x");
      }
    }
  }
}
```

如果没有加上类的声明，这个程序就无法通过编译！

C

```
class Exercise1b {
  public static void main(String [] args) {
    int x = 5;
    while ( x > 1 ) {
      x = x - 1;
      if ( x < 3) {
        System.out.println("small x");
      }
    }
  }
}
```

while 循环必须要在方法里面！

迷题解答



```
class PoolPuzzleOne {
  public static void main(String [] args) {
    int x = 0;

    while ( X < 4 ) {

      System.out.print("a");
      if ( x < 1 ) {
        System.out.print(" ");
      }
      System.out.print("n");

      if ( X > 1 ) {

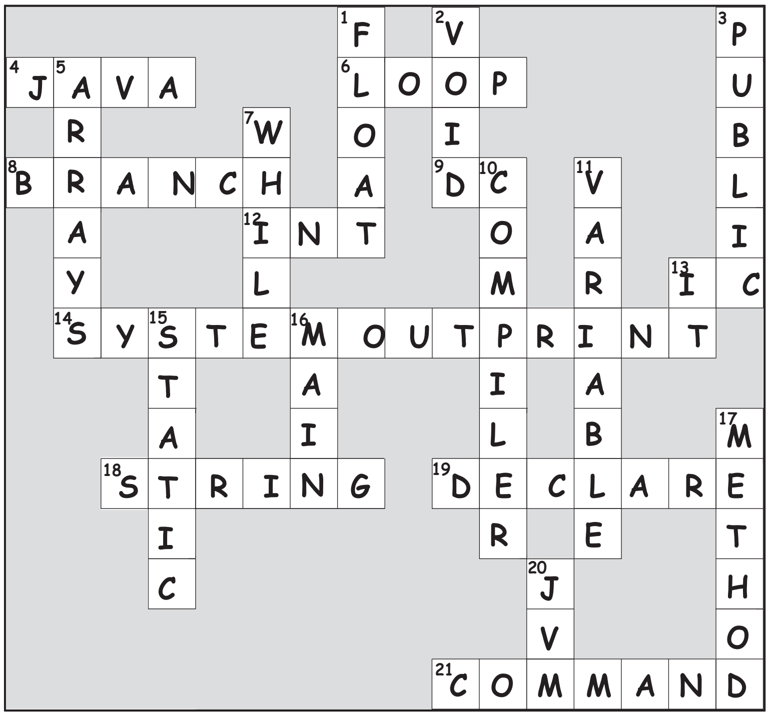
        System.out.print(" oyster");
        x = x + 2;
      }
      if ( x == 1 ) {

        System.out.print("noys");
      }
      if ( X < 1 ) {

        System.out.print("oise");
      }
      System.out.println("");

      X = X + 1;
    }
  }
}
```

```
File  Edit  Window  Help  Cheat
%java PoolPuzzleOne
a noise
annoys
an oyster
```




```
class Test {
  public static void main(String [] args) {
    int x = 0;
    int y = 0;
    while ( x < 5 ) {

      System.out.print(x + "" + y +" ");
      x = x + 1;
    }
  }
}
```

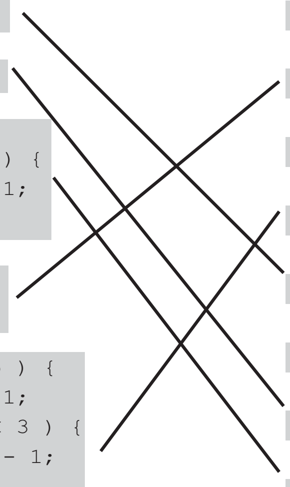
```
y = x - y;
```

```
y = y + x;
```

```
y = y + 2;
if( y > 4 ) {
  y = y - 1;
}
```

```
x = x + 1;
y = y + x;
```

```
if ( y < 5 ) {
  x = x + 1;
  if ( y < 3 ) {
    x = x - 1;
  }
}
y = y + 2;
```



22 46

11 34 59

02 14 26 38

02 14 36 48

00 11 21 32 42

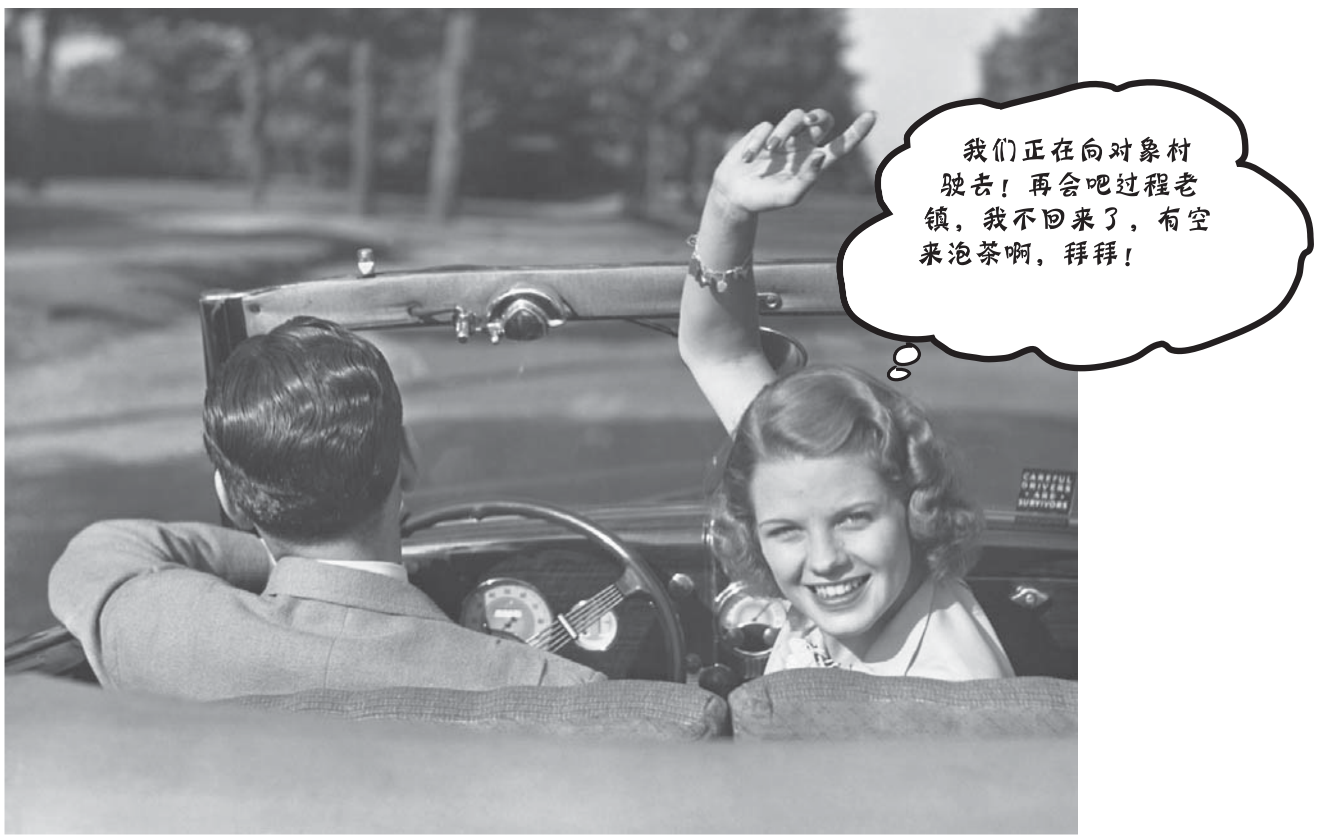
11 21 32 42 53

00 11 23 36 410

02 14 25 36 47

2 类与对象

拜访对象村



有人告诉我那里遍地都是对象。 在第1章中，我们把所有的程序代码放在main()里面。事实上，那根本就不是面向对象的做法。我们是调用到一些对象，比如String等，但是没有开发出自己设计的对象类型。所以我们要离开过程化的世界，开始建立自己的对象。我们会看到为何 Java 中的面向对象开发是如此的有趣。我们也会看到类与对象的不同，以及对象是如何让你的生活更美好（至少程序设计工作的部分会更好，但对于是否能够受到异性的青睐就不一定了）。注意：一旦进入对象村，你就不想再回头。

椅子大战

（又称：对象如何改变你的一生）

从前，有一间软件小铺，里面有两个程序员被指派去设计一个程序。

坏心的老板娘兼项目经理要求两个人比赛，赢的人可以坐上象征身分地位的Aeron™宝椅。程序开发高手阿珠跟面向对象信徒阿花两个人都认为自己赢的可能性很大。

阿珠坐在自己的座位上想着：“这个程序要执行什么动作？我会需要什么样的程序？有了，我需要rotate与playSound！”，然后她就开始进行设计的工作。

同时，阿花去倒了一杯咖啡回来，心想着：“这个程序有什么样的事物？有什么关键角色？”。她首先想到形状体（shape）。当然啦，她还会想到用户、声响等对象与点击等事件。然而这些对象早就已经建立好了，所以她只需要专注于创建形状体就行了。

接下来就是她们如何设计程序的故事以及你最想知道的答案：“谁赢了 Aeron™ 宝椅？”。





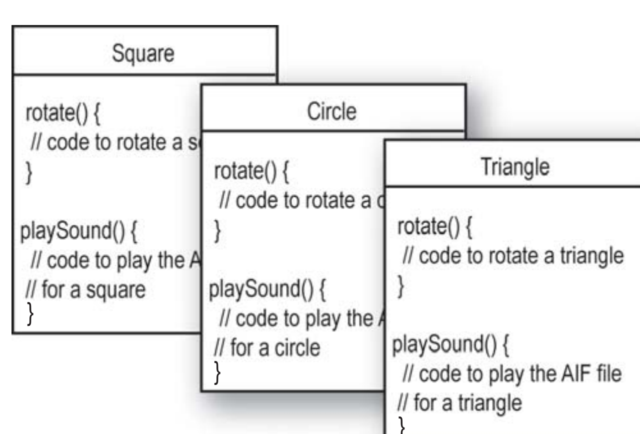
阿珠

如同以往，她准备好要开始设计重要的程序。没几下她就写出了rotate与playSound两个方法。

```
 rotate(shapeNum) {
   // 旋转360°
 }
 playSound(shapeNum) {
   // 查询播放哪个AIF文件
   // 播放
 }
```

阿花

阿花分别为3个形状各写出一个类。

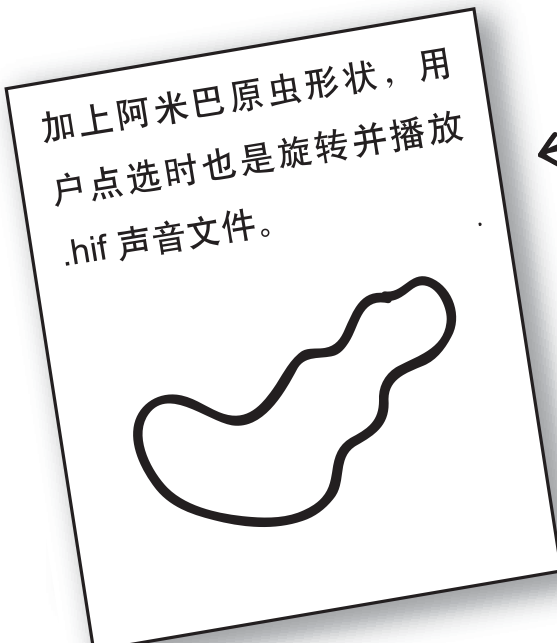


正当阿珠心想说赢定了，开始幻想坐在宝椅上接受大家投以羡慕的眼光……

等一下！老板娘说规格改了。

“OK，技术上来说阿珠赢了……大毛！你给我过来！”老板娘一边追打小孩一边说：“但是我作了一点小小的改变，对你们这种高手来说一定很简单的”。

“说呀，还要改什么？”阿珠与阿花不约而同地盯着角落的折凳看，四只手好像随时准备抄起凳子开始干活，不过想到已经忍了这么久，也不差这一次。



新加入的程序规格

阿珠

原来的rotate程序还可以用；该程序使用一个对应表来找寻特定编号的图形。但是playSound就得要修改。还有.hif文件是什么鬼东西？

```
playSound(shapeNum) {
    // 如果不是阿米巴原虫
        // 查询使用哪个AIF文件
        // 播放
    // 不然
        // 播放amoeba .hif
}
```

虽然修改幅度不大，但是她实在不想去碰已经测试过的程序代码。她应该很清楚，不管项目经理怎么保证，规格就是会不停地改。

阿花

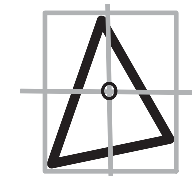
她苦笑一下，耸一耸肩，坐下来写出一个新的类。面向对象让她最喜欢的其中一点就是有时不需动到已经测试好的程序就可以达成新目标。面向对象的适应性与可扩展性让她面对修改时不会觉得太过于痛苦。

```
Amoeba
---------------
rotate() {
  // 旋转
}
playSound() {
  // 播放
}
```

阿珠悄悄地领先了阿花

但是听到坏心老板娘用失望的口吻说出“唉呀，不对呀，阿米巴原虫不是这样旋转的……”时，阿珠的脸色都变了。

原来，两个人都把旋转的部分写成这样：



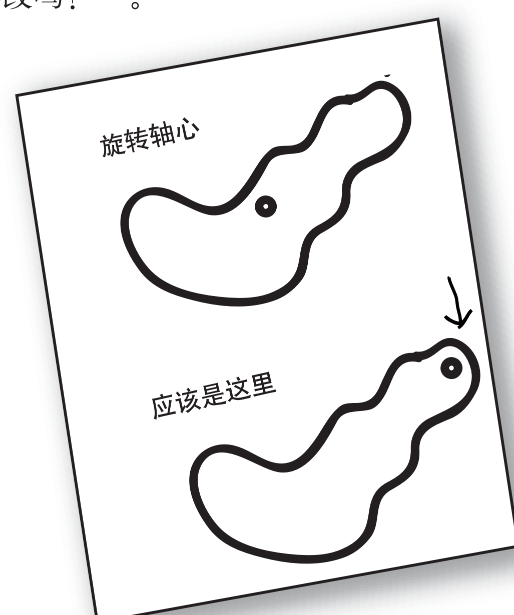
（1）找出指定形状的外接四边形。

（2）计算出四边形的中心点，以此点为轴作旋转。

但是老板娘认为阿米巴原虫应该是要绕着一端旋转，类似秒针那样。

“坏了”，阿珠心里这么想着，眼前浮现出早上烤焦的两片土司。“但我还是可以加上额外的if/else到rotate程序中硬改给阿米巴原虫，这样应该还好吧”。然而脑海中不断有个声音在提醒她：“太天真了，你以为规格不会再改吗？”。





阿珠

她想到最好还是帮rotate程序加上轴心点的参数。这样就有一堆程序要改。本来已经测试好的东西全部又得重来一遍。

```
rotate(shapeNum, xPt, yPt) {
  // 如果不是阿米巴
     // 计算中心点
     // 然后旋转
  // 否则
     // 以xPt和yPt作为旋转中心
     // 然后旋转
}
```

阿花

她修改了rotate这个方法，但不是每个都要改，只修改Amoeba这个类而已。其他已经测试、编译过的部分完全没有必要改。该类要作的修改就是加上旋转轴心点的属性（attribute）。

```
Amoeba
-----------------
int xPoint
int yPoint
rotate() {
  // 使用阿米巴的x和y
  // 执行旋转
}
playSound() {
  // 播放
}
```

所以阿花赢了，对吧？



阿珠觉得自己会赢

还早，阿珠发现阿花的方法有缺陷。并且她认为假如能够赢得胜利，老板娘的地位早晚也会被她取代，因此她得要扭转形势。

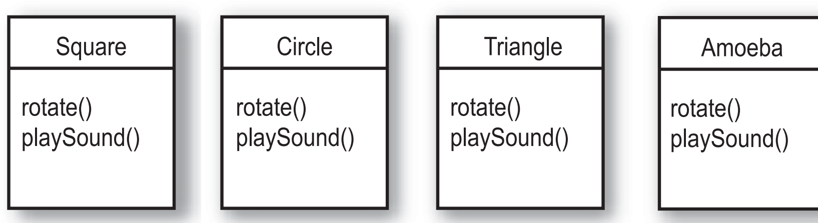
阿珠：“你有重复的程序代码！在4个Shape物体中都有rotate过程”。

阿花：“那不叫过程，那是方法好吗？还有，物体的正式名称叫做类”。

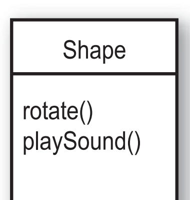
阿珠：“无所谓。你的设计有问题。这样一来你必须同时维护4个不同的rotate方法。这一点效率都没有”。

阿花：“我猜你一定没看到最终的设计。阿珠，让我告诉你什么叫做面向对象的继承（inheritance）”。

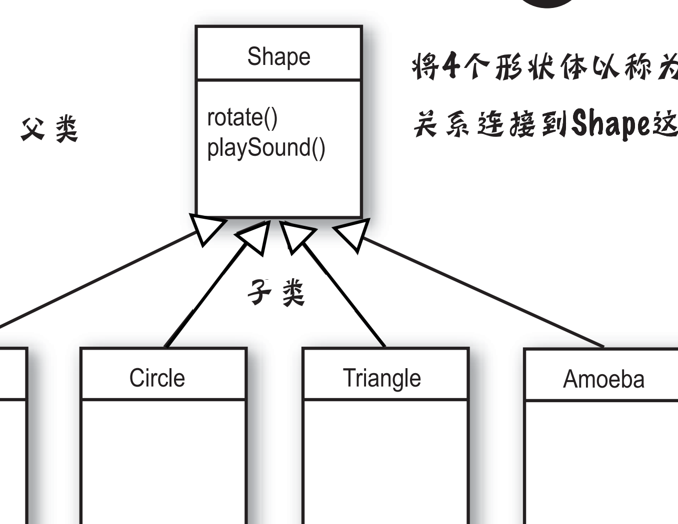
1 找出4个类中共同的部分



2 它们都是Shape，而且都有rotate与playSound，因此可以提取出新的类。



3 将4个形状体以称为继承的关系连接到Shape这个类。



这可称为“Square继承自Shape”、“Circle继承自Shape”等。rotate()与playSound()已经从次级的类中移开。

Shape是底下4个子类的父类。次级类会继承上级类的方法。也就是说，子类会自动获得父类的功能。

那阿米巴的rotate()要怎么办？

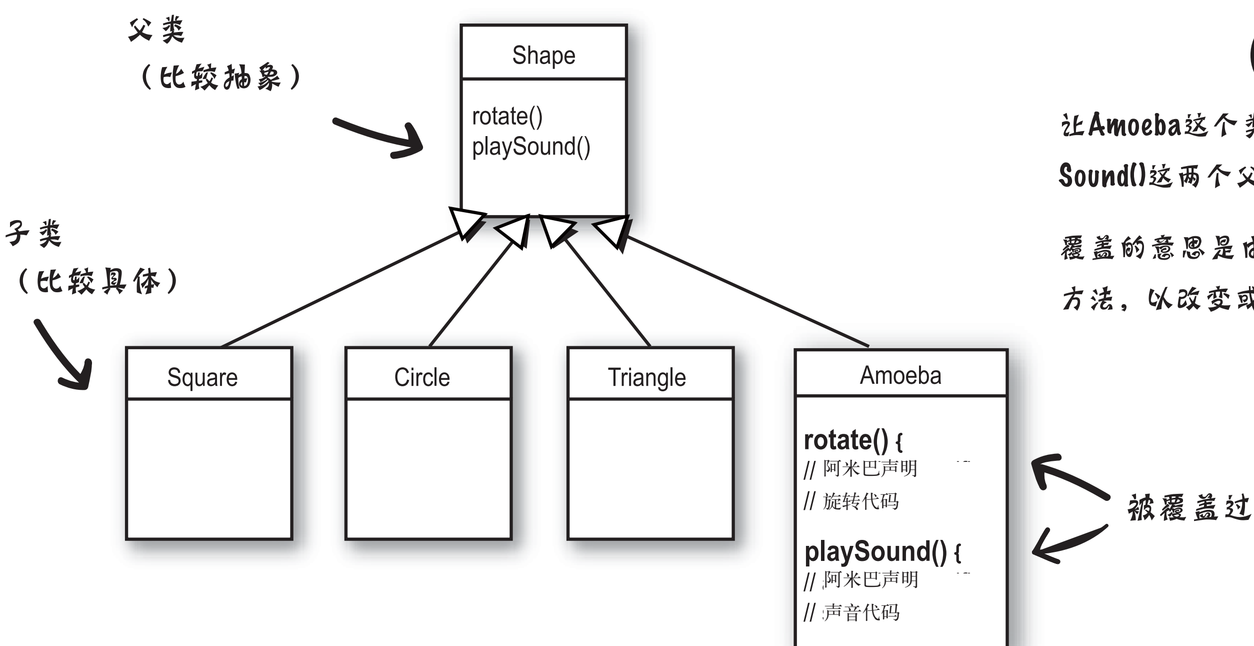
阿珠：“问题不就出在这里吗？阿米巴形状会需要完全不同的rotate与playSound程序？”



阿花：“那叫方法。”

阿珠：“如果阿米巴也是继承自Shape，那旋转的功能不就统统一样吗？”

阿花：“问得好。Amoeba这个类可以覆盖（override）Shape的方法。Java虚拟机会知道在遇到Amoeba时使用不同的rotate()。



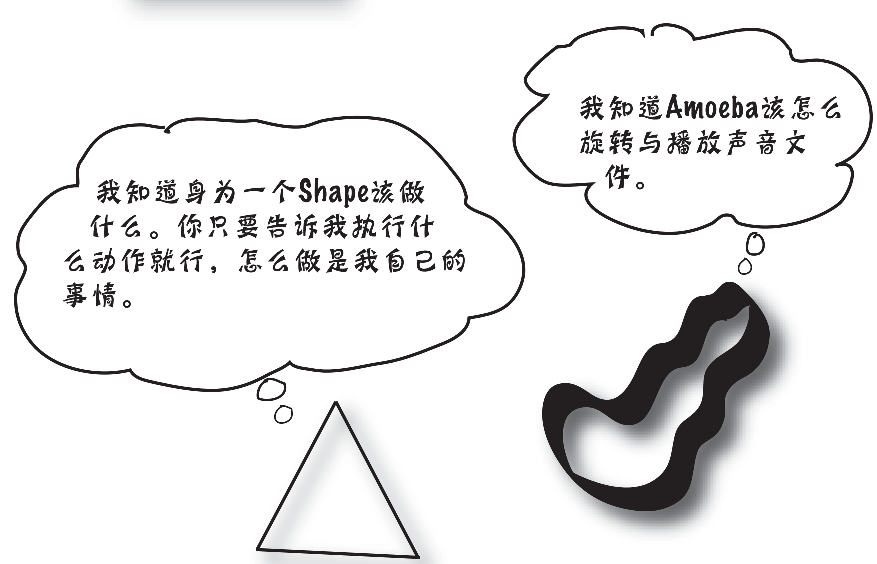
4

让Amoeba这个类去覆盖rotate()与playSound()这两个父类的方法。

覆盖的意思是由子类重新定义继承下来的方法，以改变或延伸此方法的行为。

阿珠：“你要怎么告诉Amoeba去执行某个动作？你不是得要调用这个过程……呃……方法然后叫它去旋转某个形状体吗？”

阿花：“这就是面向对象的重点。当你要旋转某个三角形的时候，程序代码会调用该三角形对象的rotate()方法。调用的一方不需知道该对象如何执行这个动作。如果要加入新的对象，只要写出该对象类型的类就行，让新的对象维持自己的行为。



剧情真是太紧张了。到底是谁赢了？



最后是阿娇获胜。

（其实有3个参赛者。附带说明：阿娇是老板娘的侄女。）

你对面向对象有什么看法？

“它帮助我用更自然的方法设计”。

——**Joy，27岁，软件架构师**

“加入新功能时不会搞乱已经写好的程序代码”。

——**Brad，32岁，程序员**

“我喜欢它将数据与操作数据的方法摆在同一个类内”。

——**Josh，22岁，啤酒品尝师**

“类可以重复运用在别的应用程序中，当写一个新类时，可以使人类有足够的扩展性，以便以后用到”。

——**Chris，39岁，项目负责人**

“我不相信Chris刚才所说的，他在近五年内没写过一行代码”。

——**Alex，35岁，Chris的工作人员**

“还有椅子大战外别的游戏吗？”。

——**Geena，26岁，程序员**

BRAIN POWER

该刺激一下神经了。

你刚刚读过面向过程程序员与面向对象程序员之间的唇枪舌战。这里面有面向对象关键概念的论述，包括了类、方法和属性。这一章接下来的部分会讨论类与对象。

用你目前所学到的概念来思考与回答下面的问题：

在设计Java的类时有哪些基本的事项要考虑？要对你自己提出哪些问题？如果要列出一份核对清单，你会列出哪些注意事项？

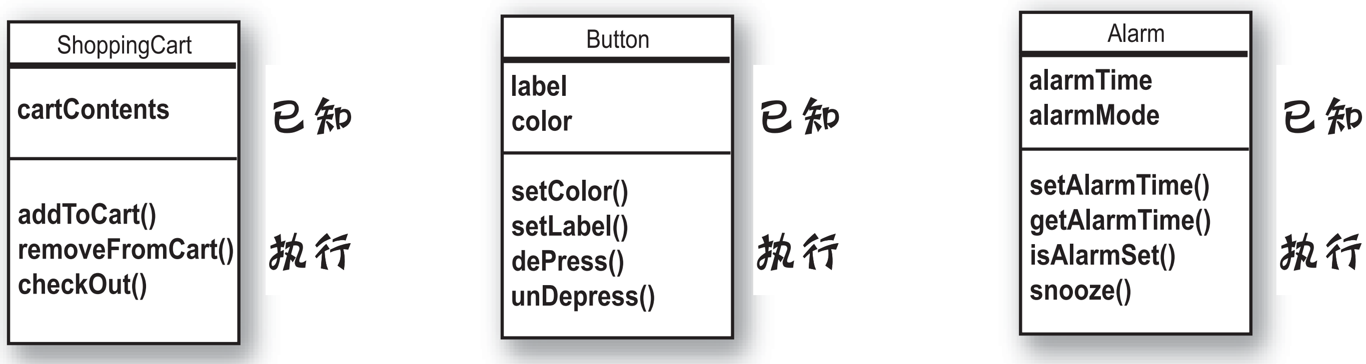


学习秘诀

如果你的思路堵塞了，试着把内容大声地念出来。说话与倾听都会用到大脑的不同位置。虽然与人交谈效果最好，但是对着宠物念也会有帮助。我家的狗狗就是这样学会JSP的。

当你在设计类时，要记得对象是靠类的模型塑造出来的。你可以这样看：

- 对象是已知的事物
- 对象会执行的动作
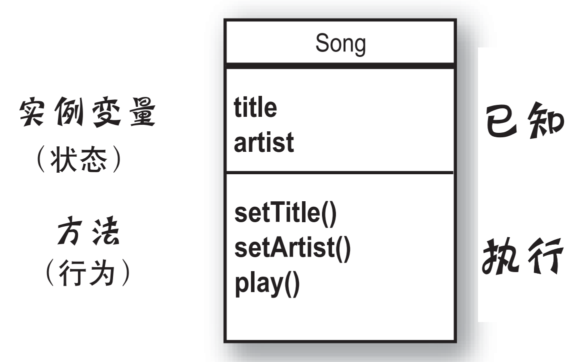


对象本身已知的事物被称为：

- 实例变量（instance variable）

对象可以执行的动作称为：

- 方法（methods）



对象本身已知的事物称为实例变量（instance variable）。它们代表对象的状态（数据），且该类型的每一个对象都会独立的拥有一份该类型的值。

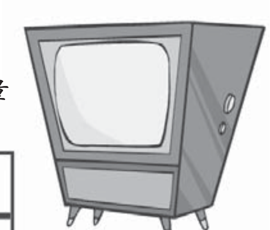
所以你也可以把对象当作为实例。

对象可以执行的动作称为方法。在设计类时，你也会设计出操作对象数据的方法。对象带有读取或操作实例变量的方法是很常见的情形。举例来说，闹钟对象会有个变量来保存响铃时间，且会有getTime()与setTime()这两个方法来存取该时间。

因此说对象带有实例变量和方法，但它们都是类设计中的一部分。

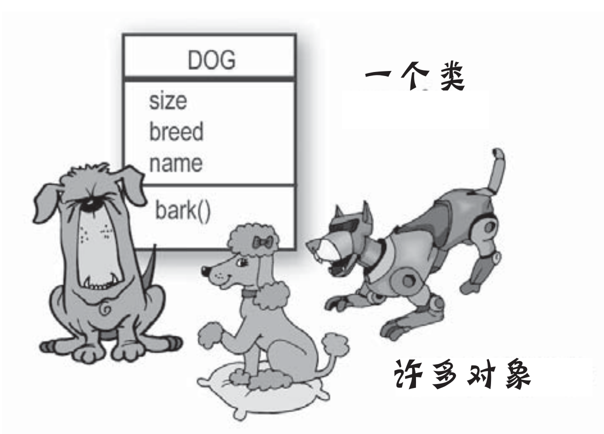


填入电视对象该有的实例变量和方法。





到底类与对象两者之间有什么不同？



类不是对象
（却是用来创建它们的模型）

类是对象的蓝图。它会告诉虚拟机如何创建某种类型的对象。根据某类创建出的对象都会有自己的实例变量。举例来说，你可以使用按钮类来创建出许多大小、颜色、文字等不同的按钮。



也可以这么说…



对象就好像通讯簿中的一笔数据

通讯簿的每张卡片都有相同的空白字段（实例变量）。填入新的联络人就如同创建新的实例（对象），填入卡片的数据代表联络人的状态。

这个卡片类上的方法就是你会对卡片作的事情：getTel()、changeAddress()、deleteCard()等。

所以每张卡能够执行相同的动作，但取出的结果应该是依每张卡片各自独立的。

创建你的第一个对象

要作出哪些东西才会运用对象呢？你需要两个类。一个是要被操作于对象的类（例如说Dog、AlarmClock和Television等），另一个是用来测试该类的类。测试用的类带有main()并且你会在其中建立与存取被测的对象。

本书后续章节的内容会有许多双类的范例。测试用的类会被命名为“受测类名称”+TestDrive。例如说要测试Bungee这个类，我们会作出一个带有main()的BungeeTestDrive。它会创建出Bungee的对象，并使用圆点（.）符号所代表的操作数来存取该对象的变量与方法。看过下面的范例后就应该更清楚了。

圆点运算符（.）

此运算符能让你存取对象的状态与行为。

```
// 建立对象
Dog d = new Dog();

// 通过操作和调用
method
d.bark();

// 通过操作数存取属性
d.size = 40;
```

1 编写类

```
class Dog {

  int size;
  String breed;
  String name;

  void bark() {
    System.out.println("Ruff! Ruff!");
  }
}
```

实例变量（指向 size、breed、name）

方法（指向 bark()）



2 编写测试用的类

```
class DogTestDrive {
   public static void main (String[] args) {
     // Dog 测试码
   }
}
```

main()（指向 main 方法）

3 在测试用的类中建立对象并存取对象的变量和方法

```
class DogTestDrive {
   public static void main (String[] args)
   {
      Dog d = new Dog();
      d.size = 40;
      d.bark();
   }
}
```

建立Dog对象（指向 Dog d = new Dog();）

存取该对象的变量（指向 d.size = 40;）

调用它的方法（指向 d.bark();）

圆点运算符（指向 d.size 和 d.bark() 中的圆点）

如果你曾经学过面向对象的话，你会知道我们并没有用到封装，第4章会讨论这个问题。

创建与测试Movie对象

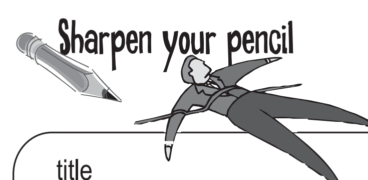


（“外星杀人对象”上映中）

```
class Movie {
  String title;
  String genre;
  int rating;

  void playIt() {
    System.out.println("Playing the movie");
  }
}

public class MovieTestDrive {
  public static void main(String[] args) {
    Movie one = new Movie();
    one.title = "Gone with the Stock";
    one.genre = "Tragic";
    one.rating = -2;
    Movie two = new Movie();
    two.title = "Lost in Cubicle Space";
    two.genre = "Comedy";
    two.rating = 5;
    two.playIt();
    Movie three = new Movie();
    three.title = "Byte Club";
    three.genre = "Tragic but ultimately uplifting";
    three.rating = 127;
  }
}
```



MOVIE
title genre rating
playIt()

MovieTestDrive这个类会创建出Movie的对象（实例）并使用圆点运算符来设定数据。MovieTestDrive也会调用其中一个对象的方法。将右方的空白处填上main()执行完毕后的对象值。

object 1
- title
- genre
- rating

object 2
- title
- genre
- rating

object 3
- title
- genre
- rating

快离开 main!

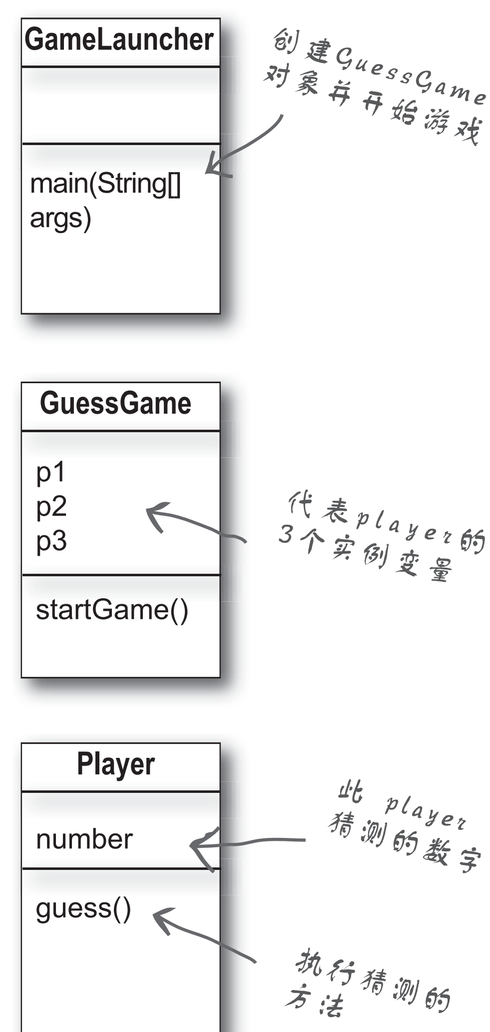
只要还呆在main()中，你就是在对象村外。呆在main()中对于一个测试用的程序来说是还好的，但对于货真价实的面向对象应用程序来说，你会需要用对象来与对象交互。

main()的两种用途：

- 测试真正的类
- 启动你的Java应用程序

真正的Java程序只会让对象与对象交互。此处所说的交互是指相互调用方法。上一页与后面的第4章会讨论在独立的TestDrive类中创建与测试其他的类。第6章会看到使用带有main()的类来启动真正的Java应用程序（创建对象并让对象之间产生交互）。

现在先给你看个真正Java应用程序会怎么做“预览”，以下是个小范例。因为我们还在学习Java的初期阶段，能够运用的技巧有限，所以程序不太优雅且无效率。你可能会思考如何将它改善，而这正是我们在后续章节会做的事。别担心看不懂某些部分；这个范例的重点在于示范对象如何与对象互动。



猜数字游戏

摘要：

这个游戏涉及到game与player两个对象。game会产生介于0～9之间的随机数字，而3个player对象会猜测该数字（你应该会觉得很无聊）。

类：

`GuessGame.class`　　`Player.class`　　`GameLauncher.class`

程序逻辑：

（1）GameLauncher 这个类带有main()方法，是应用程序的入口点。

（2）main()中会创建出GuessGame对象，并调用它的startGame()方法。

（3）startGame()方法是游戏的起点。它会创建3个player，然后挑出要猜测的随机数字。它会要求player猜测并检查结果，过程会被列出来。

```
public class GuessGame {
   Player p1;
   Player p2;
   Player p3;
```

用3个实例变量分别表示3个Player对象

```
   public void startGame() {
     p1 = new Player();
     p2 = new Player();
     p3 = new Player();
```

创建出Player对象

```
     int guessp1 = 0;
     int guessp2 = 0;
     int guessp3 = 0;
```

声明3个变量来保存是否猜中

```
     boolean p1isRight = false;
     boolean p2isRight = false;
     boolean p3isRight = false;
```

声明3个变量来保存猜测的数字

```
     int targetNumber = (int) (Math.random() * 10);
     System.out.println("I'm thinking of a number between 0 and 9...");
```

产生谜底数字

```
     while(true) {
       System.out.println("Number to guess is " + targetNumber);

       p1.guess();
       p2.guess();
       p3.guess();
```

调用Player的guess()方法

```
       guessp1 = p1.number;
       System.out.println("Player one guessed " + guessp1);

       guessp2 = p2.number;
       System.out.println("Player two guessed " + guessp2);

       guessp3 = p3.number;
       System.out.println("Player three guessed " + guessp3);
```

取得每个Player所猜测的数字并将它列出

```
       if (guessp1 == targetNumber) {
          p1isRight = true;
       }
       if (guessp2 == targetNumber) {
          p2isRight = true;
       }
       if (guessp3 == targetNumber) {
         p3isRight = true;
       }
```

检查是否猜中，若是猜中则去设定是否猜中的变量

```
       if (p1isRight || p2isRight || p3isRight) {
```

如果有一或多个猜中……
(|| 代表或运算符)

```
         System.out.println("We have a winner!");
         System.out.println("Player one got it right? " + p1isRight);
         System.out.println("Player two got it right? " + p2isRight);
         System.out.println("Player three got it right? " + p3isRight);
         System.out.println("Game is over.");
         break; // 游戏结束，中止循环

       } else {
         // 都没猜到，所以要继续下去
         System.out.println("Players will have to try again.");
       } // if/else结束
     } // 循环结束
   } // 方法结束
} // 类结束
```

不然的话就重复循环继续猜下去

运行猜数字游戏

```
public class Player {
   int number = 0;  // 要被猜的数字

     public void guess() {
     number = (int) (Math.random() * 10);
     System.out.println("I'm guessing "
                        + number);
   }
}
```

```
public class GameLauncher {
   public static void main (String[] args) {
      GuessGame game = new GuessGame();
      game.startGame();
   }
}
```



Java会拾荒

创建对象时，它会被存放在称为堆的内存区域中。不管对象如何创建都会放在此区域中。此区域并非普通的堆；它是可回收垃圾的堆（Garbage-Collectible Heap）。Java会根据对象的大小来分配内存空间。比如说15个实例变量的对象所占用的空间就可能会比只有两个实例变量的对象要大。但对象使用完毕时内存要如何回收呢？Java会主动帮你管理内存！当某个对象被Java虚拟机察觉不再会被使用到，该对象就会被标记成可回收的。如果内存开始不足，垃圾收集器就会启动来清理垃圾、回收空间，让空间能够再次被利用。后面的章节会对此机制有更多的讨论。

输出（每次执行都会不一样）

File Edit Window Help Explode

```
%java GameLauncher
I'm thinking of a number between 0 and 9...
Number to guess is 7
I'm guessing 1
I'm guessing 9
I'm guessing 9
Player one guessed 1
Player two guessed 9
Player three guessed 9
Players will have to try again.
Number to guess is 7
I'm guessing 3
I'm guessing 0
I'm guessing 9
Player one guessed 3
Player two guessed 0
Player three guessed 9
Players will have to try again.
Number to guess is 7
I'm guessing 7
I'm guessing 5
I'm guessing 0
Player one guessed 7
Player two guessed 5
Player three guessed 0
We have a winner!
Player one got it right? true
Player two got it right? false
Player three got it right? false
Game is over.
```

there are no Dumb Questions

问： **若需要全局（global）变量或方法时该如何？**

答： 在Java的面向对象概念中并没有全局变量这回事。然而实际上会有需要方法或常量（constant）可被任何的程序存取。比如说专家术语学习机中到处都在调用的random()方法或圆周率这种常数。第10章会讨论到public与static这些让方法变成类似“global”的修饰符。在任何类中的任何程序都可以存取public static的方法。任何变量只要加上public、static和final，基本上都会变成全局变量取用的常数。

问： **如果能做出全局的函数与数据，那又怎么算得上是面向对象呢？**

答： 首先要注意到任何Java中的事物都必须呆在类中。因此，pi常数或random()方法也必须定义在Math这个类中。而你必须记住这类近似全局的事物在Java中算是例外。它们是非常特殊的情况，不会有多个实例或对象。

问： **什么是Java程序？如何进行提交？**

答： Java程序是由一组类所组成，其中有一个类会带有启动用的main()方法。因此程序员必须要编写一或多个类并以此提交。若用户没有Java虚拟机则必须一并提交才能让应用程序运行起来。有数种安装程序能够让你集成包装类与不同平台使用的Java虚拟机到安装光盘上。如此就能让用户同时安装正确版本的Java虚拟机（如果之前没有安装的话）。

问： **若有成百上千的类时要如何提交？是否可以包装成类似单一应用程序的形式？**

答： 数量庞大的个别文件确实会让用户头疼。你可以把所有文件包装进依据pkzip格式来存档的Java Archive－.jar文件。在jar文件中可以引入一个简单文字格式的文字文件，它被称为manifest，里面有定义出jar中的哪一个文件带有启动应用程序的main()方法。



要点

- 面向对象设计扩展功能不需改动之前已经测试好的程序代码。
- 所有的Java程序都定义在类中。
- 类如同蓝图描述该类型的对象要如何创建。
- 对象自治；你无需在意它如何完成任务。
- 对象有已知的事物，并能执行工作。
- 对象本身已知道的事物称为实例变量，它代表对象的状态。
- 对象可执行的动作称为方法，它代表对象的行为。
- 创建类时，可能同时会需要创建独立、测试用的类。
- 类可以继承自较为抽象的父类。
- Java的程序在执行期是一组会互相交谈的对象。



我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪支程序可以编译通过。如果有问题，哪里需要修改？

A

```
class TapeDeck {

  boolean canRecord = false;

  void playTape() {
    System.out.println(“tape playing”);
  }

  void recordTape() {
    System.out.println(“tape recording”);
  }
}

class TapeDeckTestDrive {
  public static void main(String [] args) {

    t.canRecord = true;
    t.playTape();

    if (t.canRecord == true) {
      t.recordTape();

    }

  }

}
```

B

```
class DVDPlayer {

  boolean canRecord = false;

  void recordDVD() {
    System.out.println(“DVD recording”);
  }
}

class DVDPlayerTestDrive {
  public static void main(String [] args) {

    DVDPlayer d = new DVDPlayer();
    d.canRecord = true;
    d.playDVD();

    if (d.canRecord == true) {
      d.recordDVD();
    }

  }

}
```



排排看



右边是被打散的 Java 程序片段，你是否能够将它们重新排列成为可以编译与运行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
d.playSnare();
```

```
DrumKit d = new DrumKit();
```

```
boolean topHat = true;
boolean snare = true;
```

```
void playSnare() {
  System.out.println("bang bang ba-bang");
}
```

```
public static void main(String [] args) {
```

```
if (d.snare == true) {
  d.playSnare();
}
```

```
d.snare = false;
```

```
class DrumKitTestDrive {
```

```
d.playTopHat();
```

```
class DrumKit {
```

```
void playTopHat () {
  System.out.println("ding ding da-ding");
}
```

File Edit Window Help Dance

```
% java DrumKitTestDrive
bang bang ba-bang
ding ding da-ding
```



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段不能使用两次，且游泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

输出：

```
File  Edit  Window  Help  Implode
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```

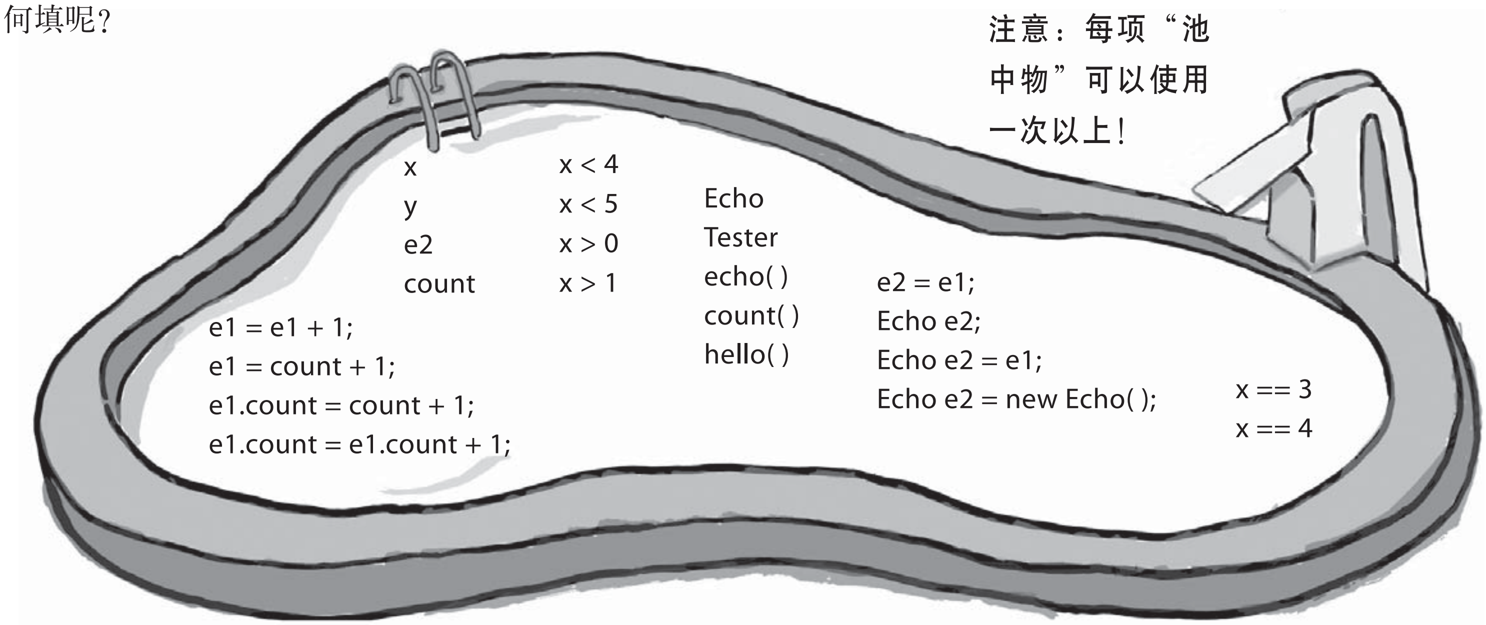
```
public class EchoTestDrive {
  public static void main(String [] args)
{
    Echo e1 = new Echo();
    _________________________
    int x = 0;
    while ( ___________ ) {
      e1.hello();
      __________________________
      if ( ____________ ) {
        e2.count = e2.count + 1;
      }
      if ( ____________ ) {
        e2.count = e2.count + e1.count;
      }
      x = x + 1;
    }
    System.out.println(e2.count);
  }
}
```

```
class ____________ {
  int _________ = 0;
  void ____________ {
    System.out.println("helloooo... ");
  }
}
```

送分题！

如果最后一行的输出不是10 而是 24，空格又应该如何填呢？

注意：每项“池中物”可以使用一次以上！





一组 Java 组件精心打扮出席化装舞会，中场时间有人提议要玩猜猜我是谁的游戏，你可以根据它们对自己的描述来猜测出提示的是哪位。规则是每个组件都得说实话，若某些提示同时对数个组件都为真的话，则将它们全部填入。第一位已经被我们猜出来了。

我是谁?



今晚出席舞会的有：

类　方法　对象　实例变量

我是由.java文件编译出来的	类
我的实例变量值可以与其他兄弟姐妹不同	
我的功能类似模板	
我喜欢执行工作	
我带有很多方法	
我代表“状态”	
我拥有很多行为	
我呆在对象中	
我生存于堆上	
我被用来创建对象实例	
我的状态可以改变	
我会声明方法	
我可以在运行期变化	



解答

排排看

```
class DrumKit {

  boolean topHat = true;
  boolean snare = true;

  void playTopHat() {
    System.out.println("ding ding da-ding");
  }

  void playSnare() {
    System.out.println("bang bang ba-bang");
  }
}

class DrumKitTestDrive {
  public static void main(String [] args) {

    DrumKit d = new DrumKit();
    d.playSnare();
    d.snare = false;
    d.playTopHat();

    if (d.snare == true) {
      d.playSnare();
    }
  }
}
```

```
File Edit Window Help Dance
% java DrumKitTestDrive
bang bang ba-bang
ding ding da-ding
```

我是编译器

A

```
class TapeDeck {
  boolean canRecord = false;
  void playTape() {
    System.out.println("tape playing");
  }
  void recordTape() {
    System.out.println("tape recording");
  }
}

class TapeDeckTestDrive {
  public static void main(String [] args) {

      TapeDeck t = new TapeDeck( );
    t.canRecord = true;
    t.playTape();

    if (t.canRecord == true) {
      t.recordTape();
    }
  }
}
```

必须要把对象建立起来！

B

```
class DVDPlayer {
  boolean canRecord = false;
  void recordDVD() {
    System.out.println("DVD recording");
  }
  void playDVD ( ) {
    System.out.println("DVD playing");
  }
}

class DVDPlayerTestDrive {
  public static void main(String [] args) {
    DVDPlayer d = new DVDPlayer();
    d.canRecord = true;
    d.playDVD();
    if (d.canRecord == true) {
      d.recordDVD();
    }
  }
}
```

必须补上playDVD()这个方法

迷题解答

泳池迷宫

```
public class EchoTestDrive {
  public static void main(String [] args) {
    Echo e1 = new Echo();
    Echo e2 = new Echo( );  // 正确答案
               - 或 -
      Echo e2 = e1;   // 也可以!
    int x = 0;
    while ( x < 4 ) {
      e1.hello();
      e1.count = e1.count + 1;
      if ( x == 3 ) {
        e2.count = e2.count + 1;
      }
      if ( x > 0 ) {
        e2.count = e2.count + e1.count;
      }
      x = x + 1;
    }
    System.out.println(e2.count);
  }
}
```

```
class Echo {
  int count = 0;
  void hello( ) {
    System.out.println("helloooo... ");
  }
}
```

```
File  Edit  Window  Help  Assimilate
%java EchoTestDrive
helloooo...
helloooo...
helloooo...
helloooo...
10
```

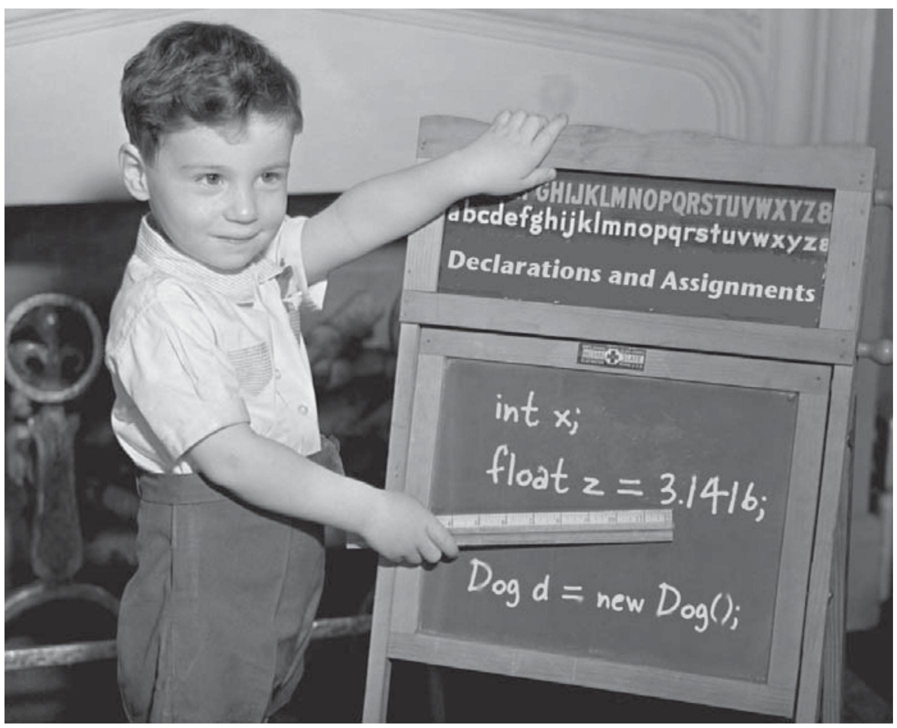
我是谁？

我是由 .java 文件编译出来的	class
我的实例变量值可以与其他兄弟姐妹不同	object
我的功能类似模板	class
我喜欢执行工作	object, method
我带有很多方法	class, object
我代表“状态”	instance variable
我拥有很多行为	object, class
我呆在对象中	method, instance variable
我生存于堆上	object
我被用来创建对象实例	class
我的状态可以改变	object, instance variable
我会声明方法	class
我可以在运行期变化	object, instance variable

注意：类（class）与对象（object）两者都说有状态与行为。它们是定义在class中的，我们并未在技术上很精确地界定对象说“有”的意义。

3 primitive主数据类型和引用

认识变量



变量有两种：primitive主数据类型和引用。到目前为止你已经在两处使用过变量——对象的状态（instance variables）与局部（local）变量（声明在方法中的变量）。稍后我们会把变量用于参数（arguments，传递给方法的值）及返回类型（执行方法所返回的值）。你已经看过被声明成primitive整数值（int类型）的变量以及声明成更为复杂如String或数组等类型的东西。但一定还有比这些东西更为复杂的事物。像是PetOwner对象会带有Dog实例变量，或者Car对象带有Engine实例变量。这一章会为你解开Java类型的迷团、探索变量的声明以及研究如何运用变量等议题。最后还会看到垃圾可回收的堆对你本周运势的影响。

声明变量



Java注重类型。它不会让你做出把长颈鹿类型变量装进兔子类型变量中这种诡异又危险的举动——如果有人对长颈鹿调用“跳跃”这个方法会发生什么样的悲剧？并且它也不会让你将浮点数类型变量放进整数类型的变量中，除非你先跟编译器确认过数字可以损失掉精确度（例如说舍去所有的小数值）。

编译器会指出大部分的问题：

```
Rabbit hopper = new Giraffe();
```

谢天谢地，这样的程序过不了编译器这关。

为了要让类型安全能够发挥作用，你必须声明所有变量的类型，指定它是个int类型或是个Dog类型。变量有两种口味：清凉的primitive主数据类型与香辣的对象引用。primitive主数据类型用来保存基本类型的值，包括整数、布尔和浮点数等。而对象引用保存的是对象的引用（嗯，这样解释很清楚吧）。

我们会先看primitive主数据类型然后再讨论对象引用真正的意义。先记住下面这条声明变量的规则：

variables must have a type

变量必须拥有类型。另一条规则是必须要有名称。

variables must have a name

```
int count;
```

类型（指向 int）　名称（指向 count）

注意：当你读到“X 类型的 Y 对象”时，类型（type）此时与类是相通的同义字。

“我要大杯的摩卡咖啡，不，中杯好了”

你可以把Java的变量想成是杯子。咖啡杯、茶杯、装满啤酒的泡沫红茶店跟鱼缸一样大的巨无霸杯、电影院贩卖爆米花用的大杯、独享杯、冠军杯、警察杯、有把手的杯以及绝对不能放进微波炉的那种镶有金属的杯。

变量就像是杯子，是一种容器，承装某些事物。

它有大小与类型。在这一章中，我们会先看过承装primitive主数据类型的变量（杯子），稍后再来看装载对象引用的杯子。我们现在以杯子的比喻来看待变量，这是比较简单的说法，后面的讨论将会越来越复杂。

primitive主数据类型如同咖啡馆的杯子，它们有不同的大小，而每种大小都有个名称，像是“小杯”、“大杯”、“重量杯”等。

你或许会在吧台上面看到展示出的杯子，可以借此选择适当的大小：



而在Java中，primitive主数据类型也有不同的大小与名称。当你在Java中声明变量的时候，必须指定它的类型。下图中的4种容器代表了Java中的4种基本整数类型。



每种杯子都可以装载数值，就像你会说“我要小杯的芒果冰沙”，你也要告诉编译器：“请给我一个int变量保存数值90”。其中有个小小的差异——你还得为杯子命名。每种primitive主数据类型变量有固定的位数（杯子大小）。存放数值的primitive主数据类型有下列6种大小：





primitive主数据类型

类型	位数	值域
boolean与char		
boolean	(Java虚拟机决定)	true或false
char	16 bits	0 ～ 65535
数值 (带正负号)		
integer		
byte	8 bits	-128～127
short	16 bits	-32768～32767
int	32 bits	-2147483648～2147483647
long	64 bits	-很大 ～ +很大
浮点数		
float	32 bits	范围规模可变
double	64 bits	范围规模可变

primitive主数据类型的声明与赋值声明：

```
int x;
x = 234;
byte b = 89;
boolean isFun = true;
double d = 3456.98;
char c = 'f';
int z = x;
boolean isPunkRock;
isPunkRock = false;
boolean powerOn;
powerOn = isFun;
long big = 3456789;
float f = 32.5f;
```

注意“f”。除非加上f，否则所有带小数点的值都会被Java当作double处理。

小心别溢出来……

要确保变量能存下所保存的值。



你无法用小杯子装大值。

好吧，其实可以，但是会损失某些信息，也就是所说的溢位。当判断到所使用的容器不足以装载时，编译器会试着防止这种情况发生。

举例来说，你无法像下面这样把int 大小的东西装进 byte 的容器中：

```
int x = 24;

byte b = x;

// 不行!
```

为什么不行呢？毕竟byte绝对装得下24这个值。你知、我知，大家都知道这回事，但对编译器来说，你正在将大物体装进小容器中，所以会有溢位的可能。就算你能够用肉眼辨别出这是安全的，但别期待编译器会看着办。

你可以用几种方式来给变量赋值：

- 在等号后面直接打出 (x=**12**, isGood = true)。
- 指派其他变量的值 (x = y)。
- 上述两种方式的组合 (x = y + **43**)。

下面粗体字部分是直接打出值的例子：

int size = **32;**	int类型的32，名称为size
char initial = **‘j’;**	char类型的‘j’，名称为initial
double d = **456.709;**	double类型的456.709，名称为d
boolean isCrazy;	只声明名称为isCrazy的boolean变量，未给值
isCrazy = **true;**	赋true值
int y = x + **456;**	名称为y的int类型变量，其值为x与456相加运算的结果

Sharpen your pencil

编译器不允许将大杯的内容放到小杯中，但反过来呢？可以。

请根据你所知道的primitive主数据类型变量类型大小，判断下列哪些赋值是合法的，哪些不合法。因为目前还没有说明所有的规则，所以你得加上自己的判断。提示：编译器在安全性的问题上比较保守。

圈出合法的述句：

```
1. int x = 34.5;
2. boolean boo = x;
3. int g = 17;
4. int y = g;
5. y = y + 10;
6. short s;
7. s = y;
8. byte b = 3;
9. byte v = b;
10. short n = 12;
11. v = n;
12. byte k = 128;
13. int p = 3 * g + y;
```

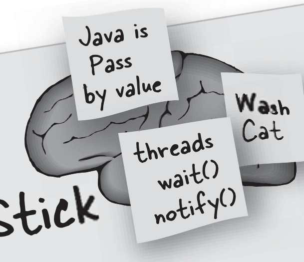
避开关键字（keyword）！

你已经知道变量需要名称和类型。

你已经知道什么是primitive主数据类型。

但是你知道命名的方法吗？很简单，你可以根据以下规则来帮助类、方法或变量命名（真正的规则在实际上更为复杂，但这些规则就能够保证安全）：

- 名称必须以字母、下划线（_）或 $ 符号开头，不能用数字开头。
- 除了第一个字符之外，后面就可以用数字。反正不要用在第一个字符就行。
- 只要符合上述两条规则，你就可以随意地命名，但还得要避开 Java 的保留字。



Make it Stick

primitive主数据类型有8种：

boolean char byte short int long float double

下列句子每个字的第一个字母刚好相同：

Be Careful! Bears Shouldn't Ingest Large Furry Dogs

如果你能自己发明类似的句子会更有帮助！

B_ C_ B_ S_ I_ L_ F_ D_

保留字是编译器要辨别的关键字。如果你想要恶搞编译器，就试试看用保留字来命名。

你已经在编写第一个main()的时候就看过几个保留字了：

```
public   static   void
```

不要使用这些字词来命名你写的东西

而primitive主数据的保留字如下：

```
boolean char byte short int long float double
```

但还有许多我们尚未讨论到的关键字。就算你不需要知道它们的意思，但还是必须清楚不能使用这些字词。现在不论在任何情况下都不要背诵下列的保留字表。留些空间给你的大脑，不然一定会忘记掉其他事情，比如你忘了把车停在哪里。学习到一种程度之后你就自然会记得了。



保留字一览表

boolean	byte	char	double	float	int	long	short	public	private
protected	abstract	final	native	static	strictfp	synchronized	transient	volatile	if
else	do	while	switch	case	default	for	break	continue	assert
class	extends	implements	import	instanceof	interface	new	package	super	this
catch	finally	try	throw	throws	return	void	const	goto	enum

这些是Java的保留字，如果你把它们用在名称上面，编译器会列出混乱的结果。

控制Dog对象

你已经知道要如何声明primitive主数据类型变量并赋值给它。但非primitive主数据类型的变量又该如何处理呢？换句话说，对象要怎么处理？

- 事实上没有对象变量这样的东西存在。
- 只有引用（reference）到对象的变量。
- 对象引用变量保存的是存取对象的方法。
- 它并不是对象的容器，而是类似指向对象的指针。或者可以说是地址。但在Java中我们不会也不该知道引用变量中实际装载的是什么，它只是用来代表单一的对象。只有Java虚拟机才会知道如何使用引用来取得该对象。

```
Dog d = new Dog();
d.bark();
```



你无法将对象装进变量中。我们通常会认为说：“我把一个String传入System.out.println()这个方法中”，或者“此方法会返回一个Dog对象”，又或是“我将新创建的Foo对象放进myFoo这个变量中”。

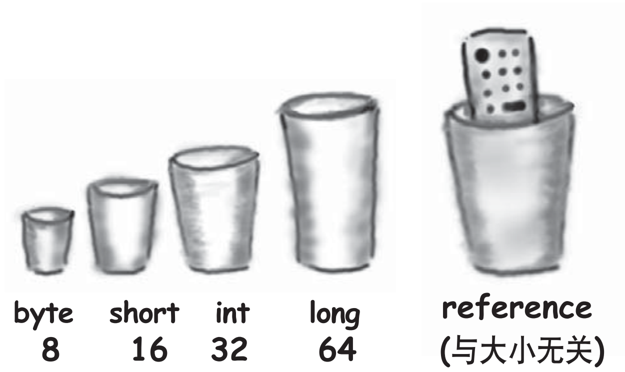
实际情况并不是这样。并没有超巨型的杯子可以放大到能够装载所有的对象。对象只会存在于可回收垃圾的堆上！（本章稍后会有更多的说明）。

虽然primitive主数据类型变量是以字节来代表实际的变量值，但对象引用变量却是以字节来表示取得对象的方法。

你会使用圆点运算符（.）来对引用变量表示：“取得圆点前面的对象，然后求出该对象在圆点后面的事物”。举例来说：

```
myDog.bark();
```

代表名为myDog的变量引用对象上的bark()。你可以把它想成遥控器与上面的按钮。



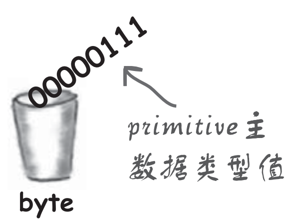
对象引用也只是个变量值

还是会有东西放进杯子中，只是引用所放进去的是遥控器。

Primitive主数据类型变量

```
byte x = 7;
```

代表数值七的字节被放进变量中(00000111).

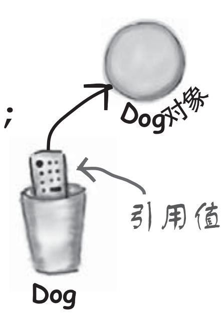


引用变量

```
Dog myDog = new Dog();
```

代表取得 Dog 对象的方法以字节形式放进变量中。

对象本身并没有放进变量中！



> 对primitive主数据类型中的变量来说，变量值就是所代表的值（如5、-26.7或‘a’）。对引用变量来说，变量值是取得特定对象的位表示法。
>
> 你不会知道或在乎某个Java虚拟机是如何实现对象引用的。它们当然有可能是指向指针的指针……就算你知道，也无法使用这些字节来实现存取对象以外其他的操作。

我们也不在乎引用变量占用多少个 0 与 1。这与 JVM 以及当时九大行星的排列有关。

对象的声明、创建与赋值有3个步骤：

```
      1          3       2
Dog myDog   =   new Dog();
```

1 声明一个引用变量

```
Dog myDog = new Dog();
```

要求Java虚拟机分配空间给引用变量，并将此变量命名为myDog。此引用变量将永远被固定为Dog类型。换句话说，它是个控制Dog的遥控器，不会是Cat或皮卡丘的遥控器。



2 创建对象

```
Dog myDog = new Dog();
```

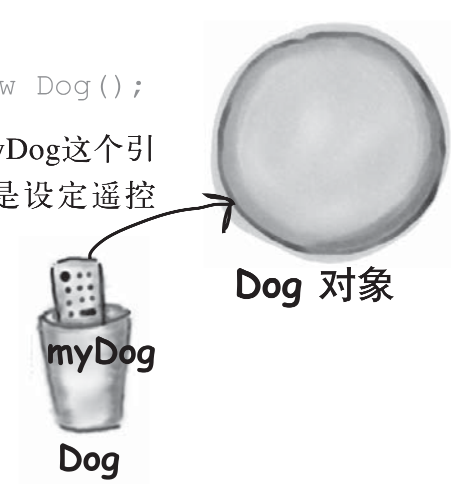
要求Java虚拟机分配堆空间给新建立的 Dog 对象（在后面，特别是第9章，我们会有更详细的讨论）。



3 连接对象和引用

```
Dog myDog = new Dog();
```

将新的Dog赋值给myDog这个引用变量。换言之就是设定遥控器。



there are no Dumb Questions

问： **引用变量有多大？**

答： 不知道。除非你跟某个Java虚拟机开发团队的人有交情，不然你是不会知道引用是如何表示的。其内部有指针，但你无法也不需要存取。若你是要讨论内存分配的问题时，最需要关心的应该是需要建立多少个对象和引用，以及对象的实际大小。

问： **既然这样，那是否意味着所有的对象引用都具有相同的大小，而不管它实际上所引用的对象大小？**

答： 是的。对于任意一个Java虚拟机来说，所有的引用大小都一样，但不同的Java虚拟机间可能会以不同的方式来表示引用，因此某个Java虚拟机的引用大小可能会大于或小于另一个Java虚拟机的引用。

问： **我可以对引用变量进行运算吗，就像C语言那样？**

答： 不行。请跟我重复念一万遍：“Java不是C”。



Java Exposed

本周的来宾：对象引用（Object Reference）

HeadFirst：请跟听众说一下对象引用的生活如何？

Reference：相当单纯，真的。我只是个遥控器，并且可以被设定来控制不同的对象。

HeadFirst：你是说在运行期间也能控制不同的对象吗？像是引用到狗对象的5分钟后又去引用皮卡丘对象？

Reference：当然不是了。被声明成什么我就是什么。如果我是个Dog遥控器，就不能指向……啊，对不起，我是说引用到Dog以外的事物。

HeadFirst：你是说你只能引用单一的Dog？

Reference：错了，我可以引用某个Dog，5分钟后又去引用另外一个Dog。只要是Dog就行，因为我可以被转换，就像重新设定遥控器一样。除非……算了。

HeadFirst：说呀，勇敢地说出来。

Reference：说完天都亮了……先简单说一下好了，如果我被标记成final的话，一旦被指派给某个Dog之后我就不能赋值给这个特定Dog之外的任何事物。也就是说被固定下来了。

HeadFirst：很好，我还真地不想现在就听到很长很长的故事。那么你能够引用到空指针吗？可以不引用任何东西吗？

Reference：是可以的，但是我不想谈这个。

HeadFirst：为什么？

Reference：这样我就会是个null，这让我很不爽。

HeadFirst：你是说没有值会让你很不爽就对了？

Reference：null也是个值。这就像你去买个万用遥控器回家，但是家里没有电视。这会让我觉得只是在浪费生命与位数，毫无意义。更糟的是，如果我是某个对象的唯一引用却又被设定成null，这意味着之后将没有其他人能够取得该对象。

HeadFirst：这会造成困扰吗？

Reference：还用说吗？我跟某个对象发生关系、有亲密的结合，然后突然间我就再也见不到这个对象，只为了它会被资源处理、垃圾回收。编写程序的人都不会考虑这么多，为什么我就不能当个primitive主数据类型，为什么让我吃到这么好吃的叉烧饭，万一以后吃不到怎么办……（访谈中断）

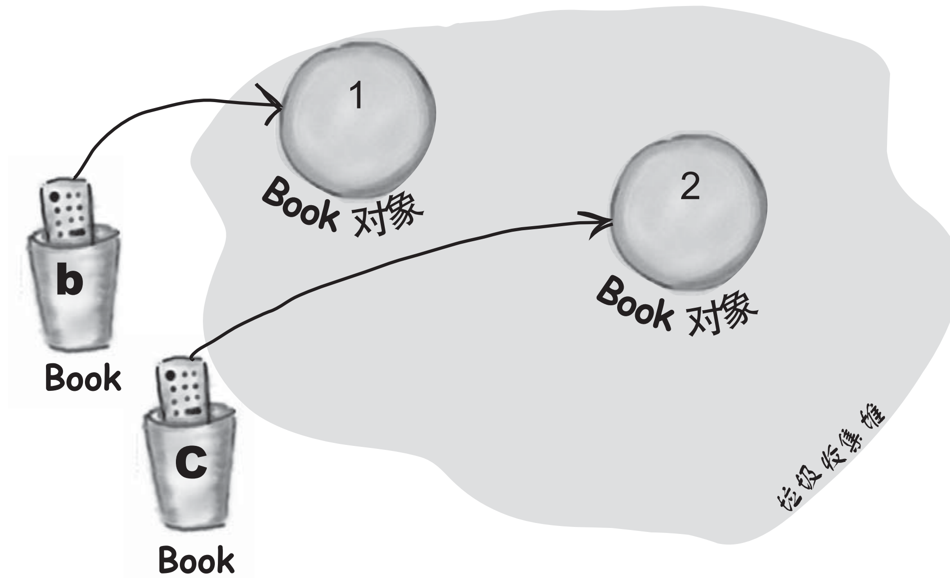
在垃圾收集堆上的生活

```
Book b = new Book();
Book c = new Book();
```

声明两个Book的引用变量并创建两个Book对象，然后将Book对象赋值给引用变量。现在这两个Book对象生活在堆上。

引用数：2

对象数：2



```
Book d = c;
```

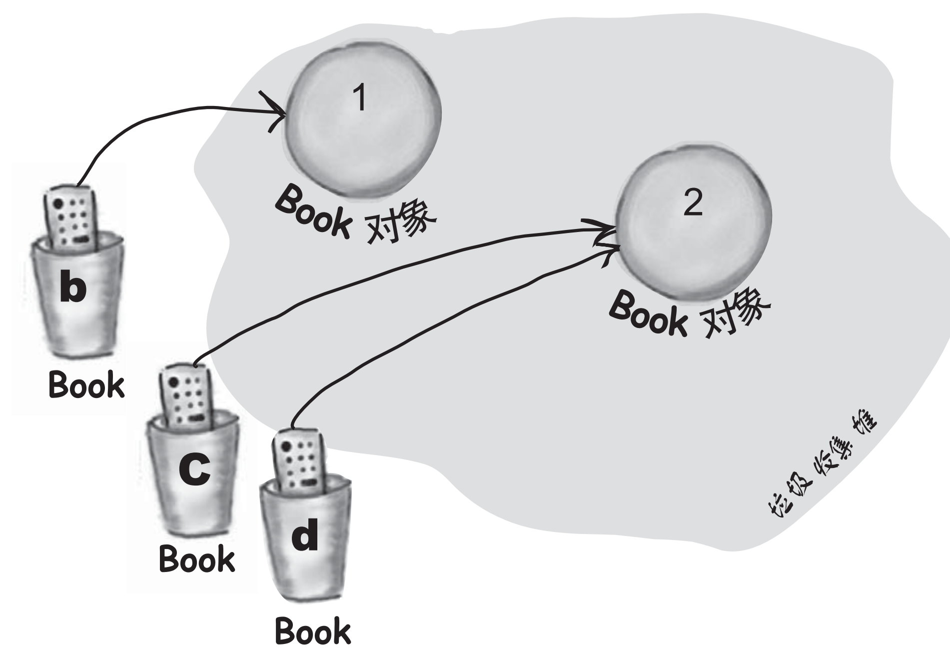
声明新的Book引用变量，但不创建新的Book对象而将变量c的值赋值给变量d。这代表“将c的字节组合拷贝给变量d”。

c与d引用到同一对象。

相同值的两份拷贝。一台电视两个遥控器。

引用数：3

对象数：2



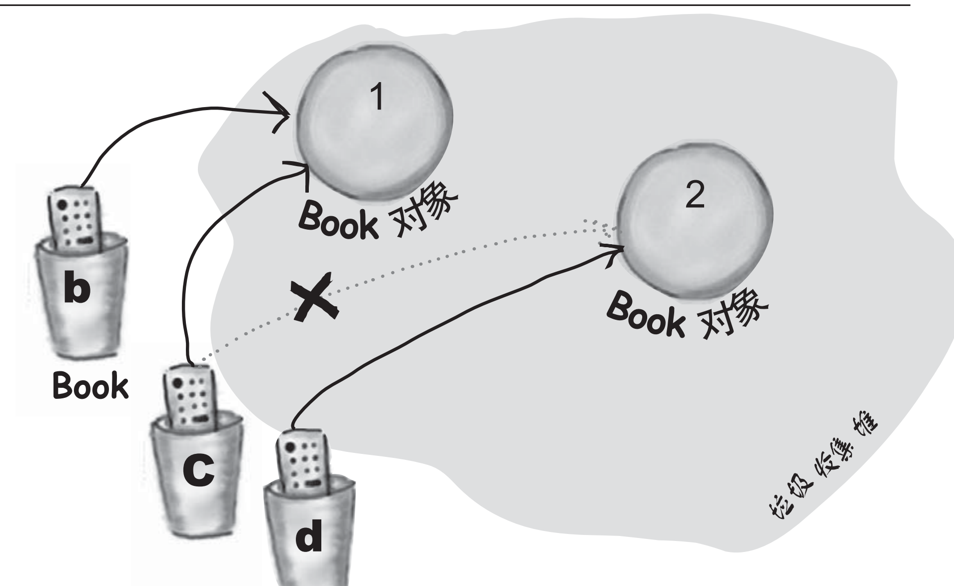
```
c = b;
```

把变量b的值赋给变量c。现在你知道这代表什么了。变量b的字节组合被拷贝一份给c。

b与c两者都引用相同的对象。

引用数：3

对象数：2



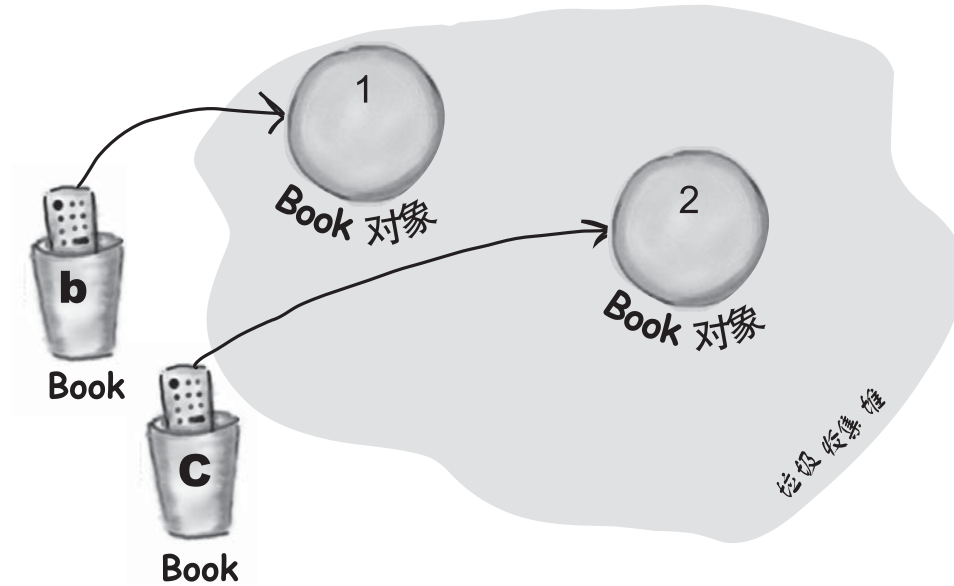
堆上的生与死

```
Book b = new Book();
Book c = new Book();
```

声明两个Book的引用变量并创建两个Book对象，然后将Book对象赋值给引用变量。现在这两个Book对象生活在堆上。

引用数：2

对象数：2




```
b = c;
```

把变量c的值赋给变量b。两者带有相同的值。

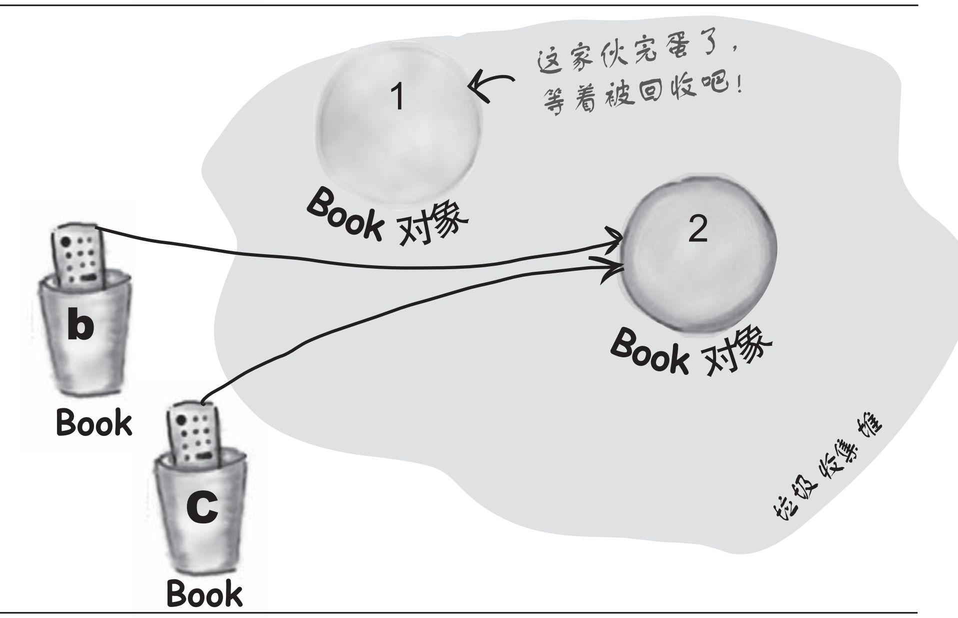
对象1被抛弃且能够作垃圾收集器(GC)。

引用数：2

对象数：2

被抛弃对象数：1

对象1已经没有引用，变成无法存取的。




```
c = null;
```

将null值赋给c。这代表它不再引用任何事物，但还是个可以被指定引用其他Book的引用变量。

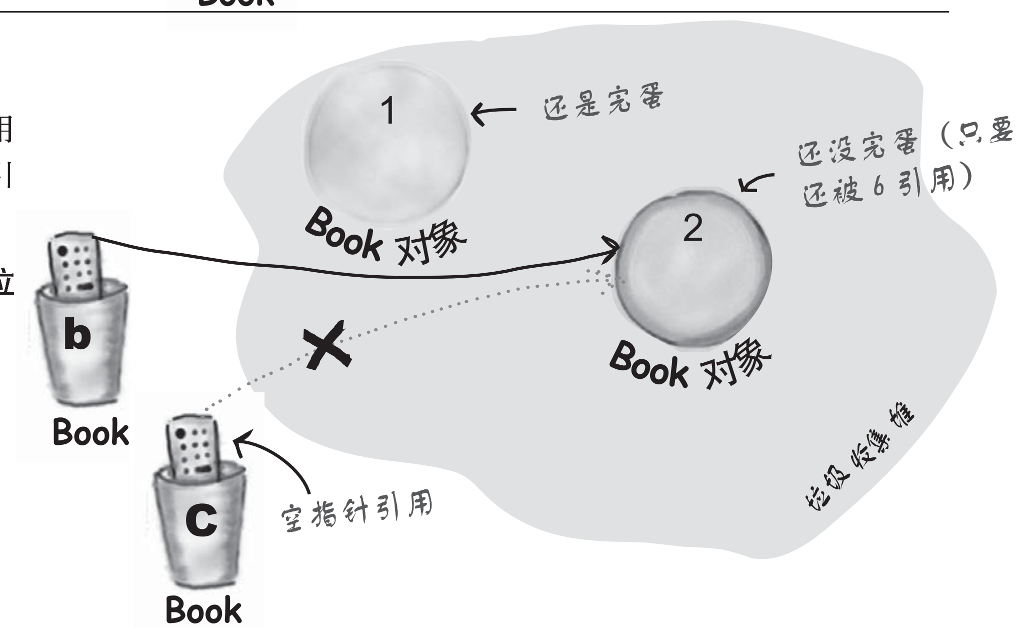
对象2还引用到，所以不能够作垃圾收集器（GC）。

作用中的引用数：1

null引用数：1

可存取对象数：1

被抛弃对象数：1



数组犹如杯架

1 声明一个int数组变量。数组变量是数组对象的遥控器。

```
int[] nums;
```

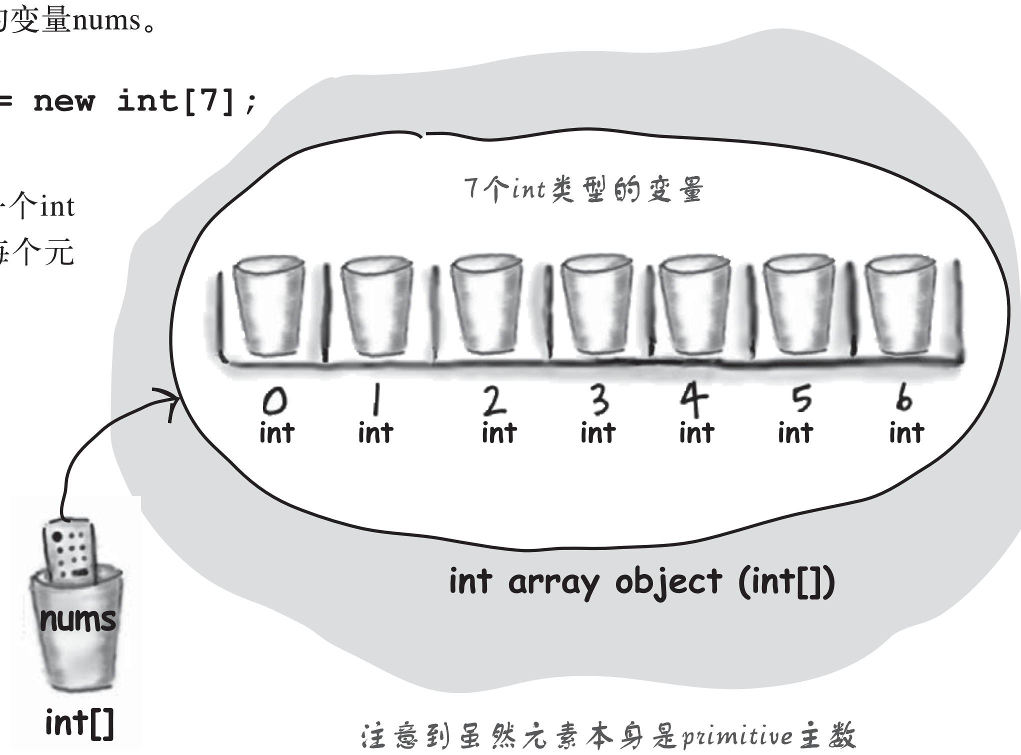
2 创建大小为7的数组，并将它赋值给之前声明为int[ ]的变量nums。

```
nums = new int[7];
```

3 赋予int数组的每一个元素一个int值。要记得在int数组中的每个元素皆为int类型的变量。

7个int类型的变量

```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```



注意到虽然元素本身是primitive主数据类型，但数组却是个对象。

数组也是对象

Java的标准函数库包含了许多复杂的数据结构，比如map、tree和set（见附录B），但如果需要快速、有序、有效率地排列元素时，数组是不错的选择。数组能够让你使用位置索引来快速、随机地存取其中的元素。

数组中的每个元素都是变量。换言之，会是8种primitive主数据类型变量中的1种，不然就是引用变量。可以放进该类型变量中的值都可以当作此类型数组的元素。所以在int类型的数组中，每个元素可以装载一个int。所以在Dog的数组中（Dog[]）每个可以装载一个Dog吗？错，要记得引用变量只会保存引用，而不是对象本身。因此Dog数组的元素持有的是Dog的遥控器。当然啦，我们还得创建Dog对象，下一页会来执行这个动作。

在上面的图中有一项要注意的：数组是个对象，不管里面放的是不是primitive主数据类型。

无论被声明来承载的是primitive主数据类型或对象引用，数组永远是对象。但你可以声明出可以装载primitive主数据类型值的数组。换句话说，数组对象可以有primitive主数据类型的元素，但数组本身绝对不会是primitive主数据类型。不管数组带有什么，它一定是对象！

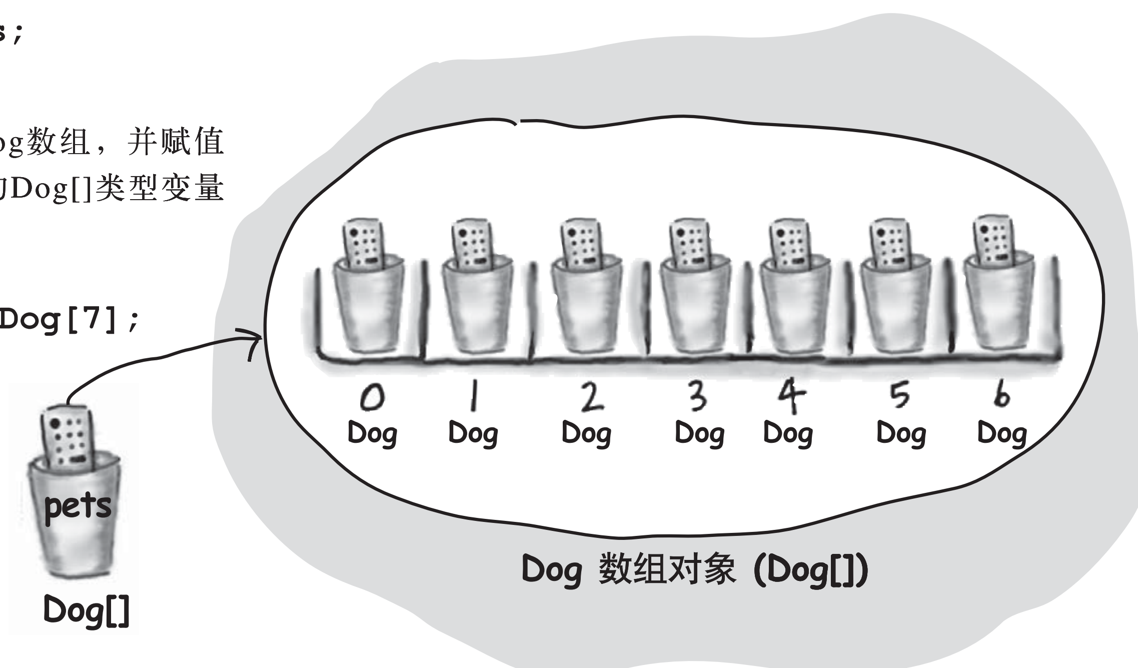
创建Dog数组

1 声明一个Dog数组变量。

```
Dog[] pets;
```

2 创建大小为7的Dog数组，并赋值给前面所声明出的Dog[]类型变量pets。

```
pets = new Dog[7];
```

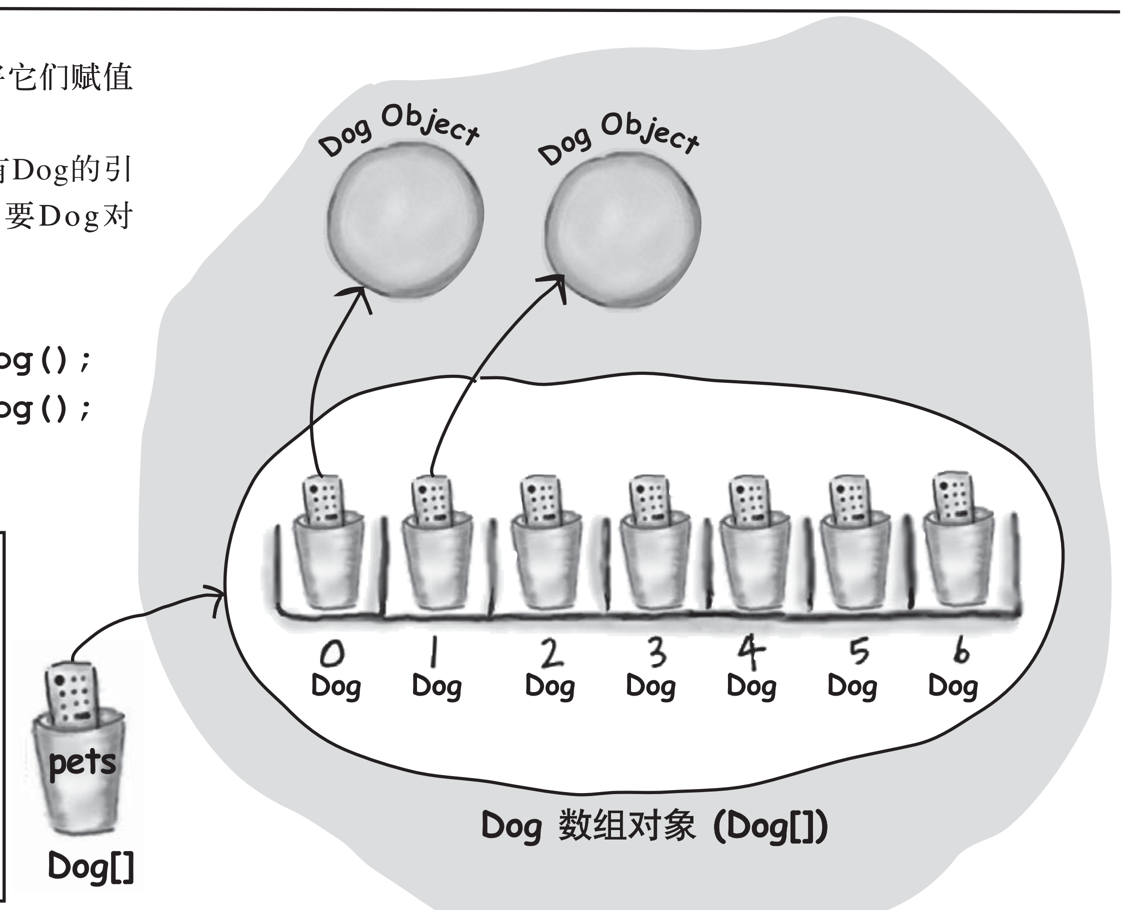


少了什么？

少了Dog！我们虽然有了对Dog的引用，但缺少实际的Dog对象！

3 创建新的Dog对象并将它们赋值给数组的元素。
记得Dog数组中只带有Dog的引用变量。我们还是需要Dog对象！

```
pets[0] = new Dog();
pets[1] = new Dog();
```



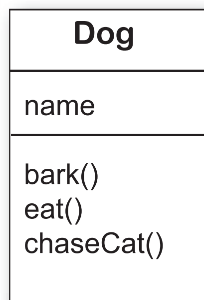
Sharpen your pencil

pets[2]目前的值是什么？____________

如何指派pets[3]引用到既有的Dog对象？

____





控制Dog

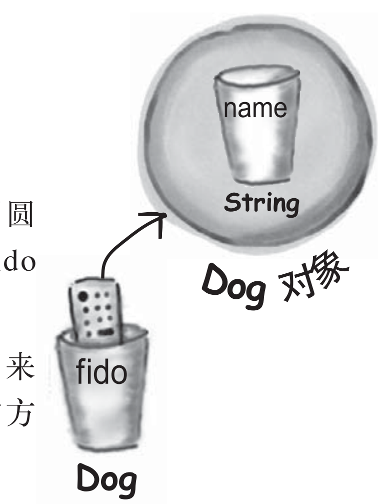
（通过引用变量）

```
Dog fido = new Dog();
fido.name = "Fido";
```

我们创建出Dog对象并使用圆点运算符来操作引用变量fido并存取它的name变量。

我们可以运用fido这个引用来让Dog执行bark()或其他的方法。

```
fido.bark();
fido.chaseCat();
```



Java注重类型

一旦数组被声明出来，你就只能装入所声明类型的元素。

举例来说，你不能将Cat放到Dog数组中（如果有人尝试要让数组中的每个元素都汪汪叫一次会出现什么状况？）。double也不能放进int数组中。但是你可以将byte放进int的数组中，因为byte可以放进int尺寸的杯子中。这被称为隐含展开（implicit widening，稍后会有更多的说明，现在只需要注意编译器会根据数组所声明的类型来防止错误的类型）。

如何存取Dog数组中的Dog？

我们知道可以通过圆点运算符来存取Dog的实例变量与方法，但数组呢？

我们对数组的操作可以不需要变量名称。只需要数组索引（位置）就可以操作特定对象了：

```
Dog[] myDogs = new Dog[3];
myDogs[0] = new Dog();
myDogs[0].name = "Fido";
myDogs[0].bark();
```

*此处的说明还未运用到封装的概念，我们会在第4章加以讨论。

```
class Dog {
  String name;
  public static void main (String[] args) {
    // 创建Dog对象
    Dog dog1 = new Dog();
    dog1.bark();
    dog1.name = "Bart";

    // 创建Dog数组
    Dog[] myDogs = new Dog[3];
    // 关门放狗
    myDogs[0] = new Dog();
    myDogs[1] = new Dog();
    myDogs[2] = dog1;

    // 通过数组引用存取Dog
    myDogs[0].name = "Fred";
    myDogs[1].name = "Marge";

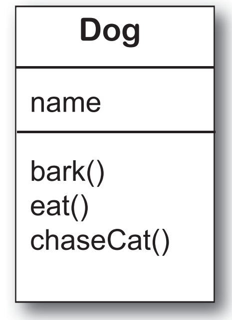
    // myDog[2]的名字是?
    System.out.print("last dog's name is ");
    System.out.println(myDogs[2].name);

    // 逐个对Dog执行bark()
    int x = 0;
    while(x < myDogs.length) {
      myDogs[x].bark();
      x = x + 1;
    }
  }

  public void bark() {
    System.out.println(name + " says Ruff!");
  }
  public void eat() {  }
  public void chaseCat() {  }
}
```

数组有个称为length的变量能够返回元素的数目

Dog的范例



输出

```
File  Edit  Window  Help  Howl
%java Dog
null says Ruff!
last dog's name is Bart
Fred says Ruff!
Marge says Ruff!
Bart says Ruff!
```

要点

- 变量有两种：primitive主数据类型和引用
- 变量的声明必须有类型和名称。
- primitive主数据类型变量值是该值的字节所表示的。
- 引用变量的值代表位于堆之对象的存取方法。
- 引用变量如同遥控器，对引用变量使用圆点运算符可以如同按下遥控器按钮般地存取它的方法或实例变量。
- 没有引用到任何对象的引用变量的值为null值。
- 数组一定是个对象，不管所声明的元素是否为primitive主数据类型，并且没有primitive主数据类型的数组，只有装载primitive主数据类型的数组。



我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪支程序可以编译过关。如果有问题，哪里要修改？

A

```
class Books {
  String title;
  String author;
}

class BooksTestDrive {
  public static void main(String [] args) {

    Books [] myBooks = new Books[3];
    int x = 0;
    myBooks[0].title = “The Grapes of Java”;
    myBooks[1].title = “The Java Gatsby”;
    myBooks[2].title = “The Java Cookbook”;
    myBooks[0].author = “bob”;
    myBooks[1].author = “sue”;
    myBooks[2].author = “ian”;

    while (x < 3) {
      System.out.print(myBooks[x].title);
      System.out.print(“ by “);
      System.out.println(myBooks[x].author);
      x = x + 1;
    }
  }
}
```

B

```
class Hobbits {

  String name;

  public static void main(String [] args) {

    Hobbits [] h = new Hobbits[3];
    int z = 0;

    while (z < 4) {
      z = z + 1;
      h[z] = new Hobbits();
      h[z].name = “bilbo”;
      if (z == 1) {
        h[z].name = “frodo”;
      }
      if (z == 2) {
        h[z].name = “sam”;
      }
      System.out.print(h[z].name + “ is a “);
      System.out.println(“good Hobbit name”);
    }
  }
}
```



排排看



右边是被打散的Java程序片段，你是否能够将它们重新排列以成为可以编译与运行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
int y = 0;
```

```
ref = index[y];
```

```
islands[0] = "Bermuda";
islands[1] = "Fiji";
islands[2] = "Azores";
islands[3] = "Cozumel";
```

```
int ref;
while (y < 4) {
```

```
System.out.println(islands[ref]);
```

```
index[0] = 1;
index[1] = 3;
index[2] = 0;
index[3] = 2;
```

```
String [] islands = new String[4];
```

```
System.out.print("island = ");
```

```
int [] index = new int[4];
```

```
y = y + 1;
```

```
class TestArrays {

  public static void main(String [] args) {
```

```
File Edit Window Help Bikini
% java TestArrays
island = Fiji
island = Cozumel
island = Bermuda
island = Azores
```



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

输出：

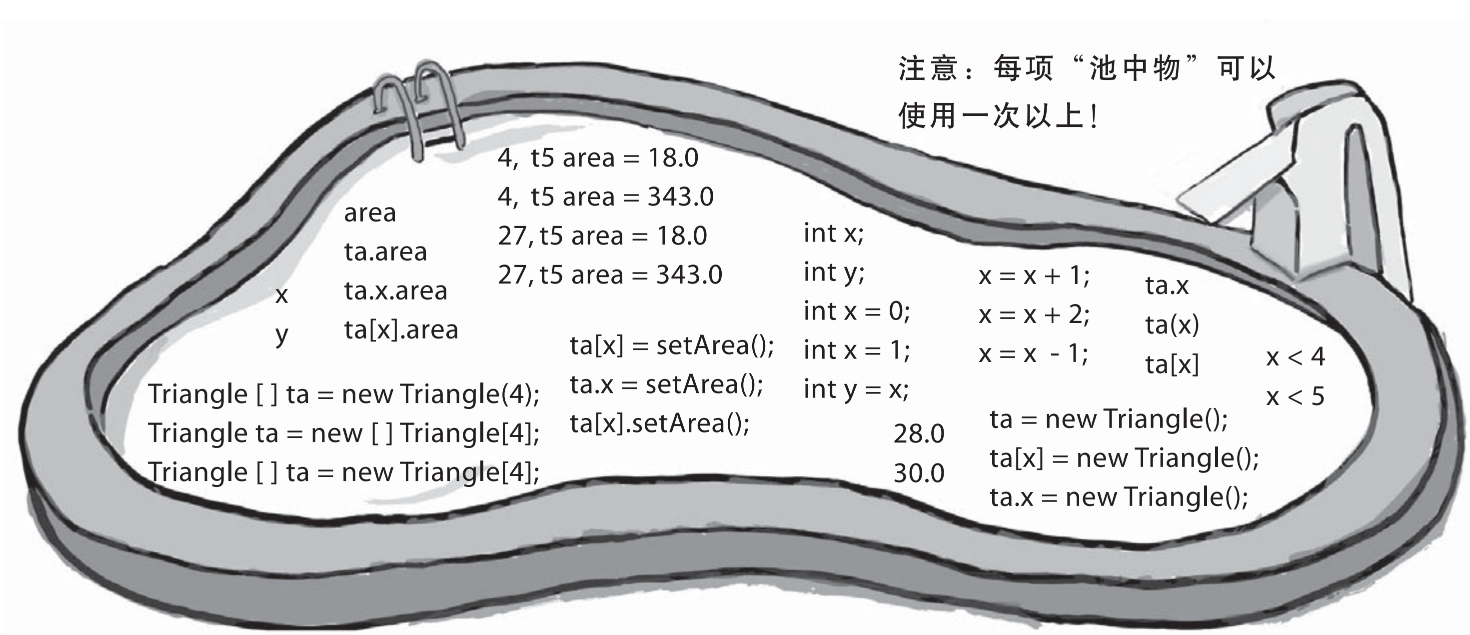
```
File  Edit  Window  Help  Bermuda
%java Triangle
triangle 0, area = 4.0
triangle 1, area = 10.0
triangle 2, area = 18.0
triangle 3, area = ____
y = ______________________
```

加分题！

从池中找出可以填在输出空格部分的片段。

```
class Triangle {
  double area;
  int height;
  int length;
  public static void main(String [] args) {
    ____________
    ______________________
    while ( __________ ) {
      __________________________
      ________.height = (x + 1) * 2;
      ________.length = x + 4;
      __________________________
      System.out.print("triangle "+x+", area");
      System.out.println(" = " + _______.area);
      ________________
    }
    ______________
    x = 27;
    Triangle t5 = ta[2];
    ta[2].area = 343;
    System.out.print("y = " + y);
    System.out.println(", t5 area = "+ t5.area);
  }
  void setArea() {
    ____________ = (height * length) / 2;
  }
}
```

（有时我们会为了节省空间而没有采用分离的测试类）





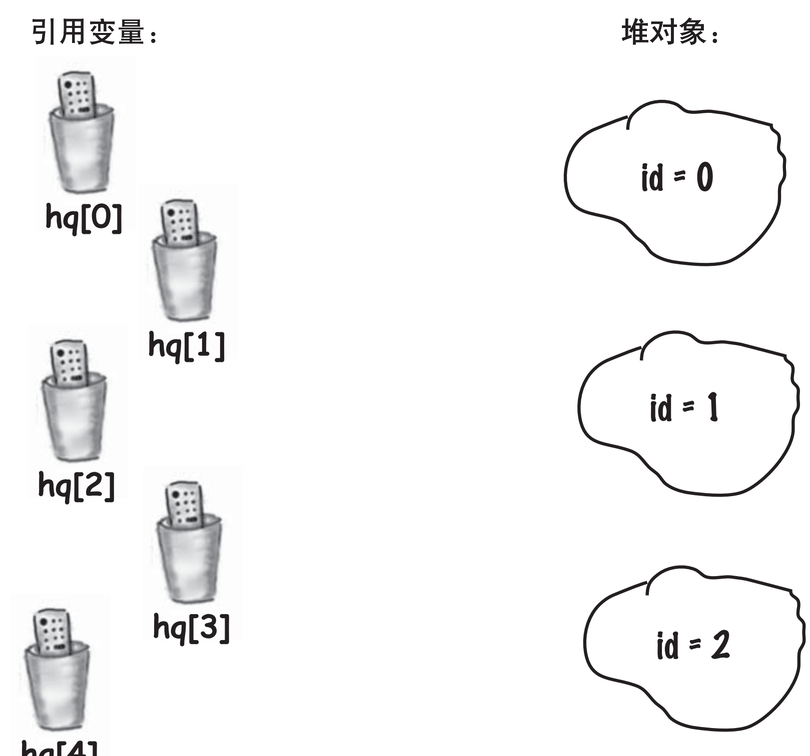
连连看

右边有一段Java小程序。执行到“// do stuff”这一行时已经创建一些对象与引用变量。你的工作是要判别哪个引用变量引用到哪个对象。引用变量不一定用到，对象也可能会被多个变量引用。对有引用关系的对象画线连接起来。

提示：你可以参考第55页与第56页的做法。

```
class HeapQuiz {
  int id = 0;
  public static void main(String [] args) {
    int x = 0;
    HeapQuiz [ ] hq = new HeapQuiz[5];
    while ( x < 3 ) {
      hq[x] = new HeapQuiz();
      hq[x].id = x;
      x = x + 1;
    }
    hq[3] = hq[1];
    hq[4] = hq[1];
    hq[3] = null;
    hq[4] = hq[0];
    hq[0] = hq[3];
    hq[3] = hq[2];
    hq[2] = hq[0];
    // do stuff
  }
}
```

将有引用关系的变量与对象画线连接起来。





阿强少年事件簿：在密室中消失的引用

在一个风雨交加的夜晚，业务部的阿美走进工程部“巡逻”。她知道程序员都还在加班，因为是她要求他们留下来赶一个程序的。她得交给客户一个移动电话用的Java通讯簿管理系统。每个人都知道移动电话的堆内存就跟阿美看得上眼的男人一样少的可怜。就在她突然走向白板时，喧闹的办公室马上就安静下来，因为大家知道她又要增加功能了。她很快在白板上画好新方法的概要图并放下笔来慢慢地扫视全场说到：“帅哥们，来吧，能给我在内存上最有效率方法的人明天就能跟我去夏威夷……帮客户安装程序”。



第二天早上，天气晴朗，鸟语花香，阿美穿着夏威夷草裙溜进工程部。“早安啊，帅哥们”，她满脸笑容地说，“谁要跟我远走高飞呢？”。暗恋阿美已久的阿强第一个跳到白板前准备展示他熬夜出来的成果。阿美说：“先让我看你怎么处理联络人对象的更新”。阿强很快就把程序展示出来：

```
Contact [] ca = new Contact[10];
 while ( x < 10 ) {    // 创建10个contact对象
   ca[x] = new Contact();
   x = x + 1;
 }
 // 执行其余复杂的更新工作
```

“这就是我写的方法”阿强显然对这个方法很满意。接着小明也跳出来，他对阿强说“你不觉得你的写法有点问题吗？”回头又对阿美说“宝贝，看完程序我们就走好吗？”：

```
Contact refc;
while ( x < 10 ) {    // 创建10个contact对象
  refc = new Contact();
  x = x + 1;
}
// 执行其余复杂的更新工作
```

“这样写才可以省下引用变量用的宝贵内存啊，学着点……”，小明以胜利者的姿态对着阿强说：“等小孩满月时一定要来啊”。阿美却不这么认为：“小明，你等下辈子吧，阿强我们走，登机前还可以先去喝个饮料……”边说边拉着阿强往等在公司门口的接送车走去。

为什么阿美选择了阿强而不是内存耗用比较少的小明？最后阿强会得逞吗？小明还有什么办法可以从中破坏两人的感情呢？



解答

排排看：

```
class TestArrays {
  public static void main(String [] args) {
    int [] index = new int[4];
    index[0] = 1;
    index[1] = 3;
    index[2] = 0;
    index[3] = 2;
    String [] islands = new String[4];
    islands[0] = "Bermuda";
    islands[1] = "Fiji";
    islands[2] = "Azores";
    islands[3] = "Cozumel";
    int y = 0;
    int ref;
    while (y < 4) {
      ref = index[y];
      System.out.print("island = ");
      System.out.println(islands[ref]);
      y = y + 1;
    }
  }
}
```

File Edit Window Help Bikini

```
% java TestArrays
island = Fiji
island = Cozumel
island = Bermuda
island = Azores
```

A

```
class Books {
  String title;
  String author;
}
class BooksTestDrive {
  public static void main(String [] args) {
    Books [] myBooks = new Books[3];
    int x = 0;
    myBooks[0] = new Books();
    myBooks[1] = new Books();
    myBooks[2] = new Books();
    myBooks[0].title = "The Grapes of Java";
    myBooks[1].title = "The Java Gatsby";
    myBooks[2].title = "The Java Cookbook";
    myBooks[0].author = "bob";
    myBooks[1].author = "sue";
    myBooks[2].author = "ian";
    while (x < 3) {
      System.out.print(myBooks[x].title);
      System.out.print(" by ");
      System.out.println(myBooks[x].author);
      x = x + 1;
    }
  }
}
```

记得创建Books对象

B

```
class Hobbits {
  String name;
  public static void main(String [] args) {
    Hobbits [] h = new Hobbits[3];
        int z = -1;
        while (z < 2) {
      z = z + 1;
      h[z] = new Hobbits();
      h[z].name = "bilbo";
      if (z == 1) {
        h[z].name = "frodo";
      }
      if (z == 2) {
        h[z].name = "sam";
      }
      System.out.print(h[z].name + " is a ");
      System.out.println("good Hobbit name");
    }
  }
}
```

记得数组由0开始

迷宫解答

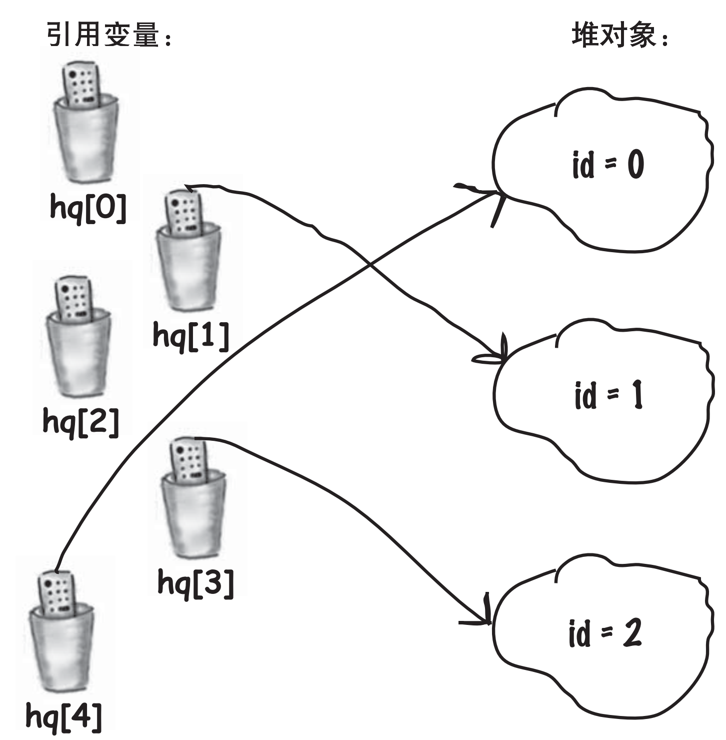
```
class Triangle {
  double area;
  int height;
  int length;
  public static void main(String [] args) {
    int x = 0;
    Triangle [ ] ta = new Triangle[4];
    while ( x < 4 ) {
      ta[x] = new Triangle();
      ta[x].height = (x + 1) * 2;
      ta[x].length = x + 4;
      ta[x].setArea();
      System.out.print("triangle "+x+", area");
      System.out.println(" = " + ta[x].area);
      x = x + 1;
    }
    int y = x;
    x = 27;
    Triangle t5 = ta[2];
    ta[2].area = 343;
    System.out.print("y = " + y);
    System.out.println(", t5 area = "+ t5.area);
  }
  void setArea() {
    area = (height * length) / 2;
  }
}
```

```
File Edit Window Help Bermuda
%java Triangle
triangle 0, area = 4.0
triangle 1, area = 10.0
triangle 2, area = 18.0
triangle 3, area = 28.0
y = 4, t5 area = 343
```

在密室中消失的引用

因为阿美看出来小明的方法有个重大缺陷。小明所占用的内存是比较少没错，但是除了最后一个Contact对象外其他的都没有办法存取。因为从头到尾只有一个引用变量，此变量最后只能引用到最新建立出的对象。因此小明的程序代码根本不能用。

（至于后来在夏威夷发生了什么事，结果又会变得怎么样，阿美是否会出车祸丧失记忆，阿强是不是私生子，小明会不会被外星人掳走，以上这些问题连我们也没有答案……）



4 方法操作实例变量

对象的行为



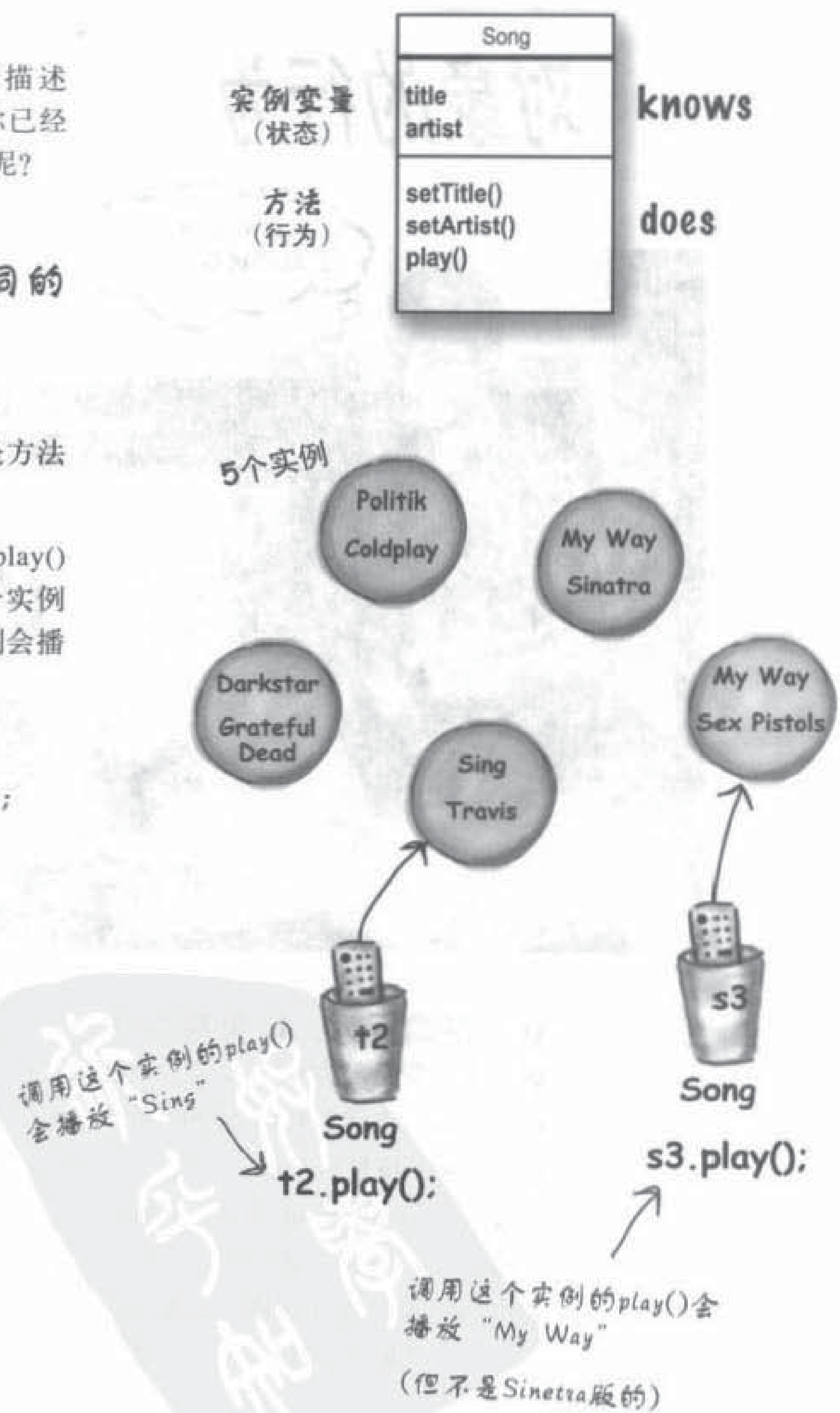
危险动作，请勿模仿

（除非你有自残的念头）

状态影响行为，行为影响状态。我们已经知道对象有状态和行为两种属性，分别由实例变量与方法来表示。但我们还没有看到两者之间的关联。我们已经知道类的每个实例（也就是特定类型的每个对象）可以维持自己的实例变量。某个Dog的名称是“Fido”质量有20kg。另外一个Dog名称为“Killer”，质量有3kg。如果Dog这个类带有一个makeNoise()方法，你觉得哪一只的吠声会比较低沉呢？（假设呜呜两声也算吠的话）。幸好这就是面向对象的重点——行为会依据状态来决定。换句话说，方法会使用到实例变量的值。比如说：“如果狗的质量超过8kg，就发出呜呜的声音，否则……”或是“加2kg”。让我们奔向夕阳、改变状态吧！

记住：类所描述的是对象知道什么与执行什么？

类是对象的蓝图。在编写类时，你是在描述Java虚拟机应该如何制作该类型的对象。你已经知道每个对象有独立的实例变量值。但方法呢？



同一类型的每个对象能够有不同的方法行为吗？

嗯……差不多*。

任一类的每个实例都带有相同的方法，但是方法可以根据实例变量的值来表现不同的行为。

Song这个类有title与artist这两个实例变量。play()会播放title值所表示的歌曲。所以调用某个实例的play()可能会播放"Politik"而另一个实例会播放"Darkstar"。然而方法却是相同的：

```
void play() {
    soundPlayer.playSound(title);
}
```

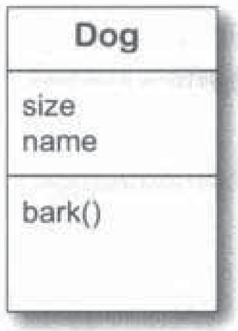
```
Song t2 = new Song();
t2.setArtist("Travis");
t2.setTitle("Sing");
Song s3 = new Song();
s3.setArtist("Sex Pistols");
s3.setTitle("My Way");
```

*无懈可击的回答！

大小影响叫声

小型犬的叫声与大型犬不同。

Dog这个类有个称为size的实例变量，bark()会用它来决定使用哪一种声音。




```
class Dog {
  int size;
  String name;

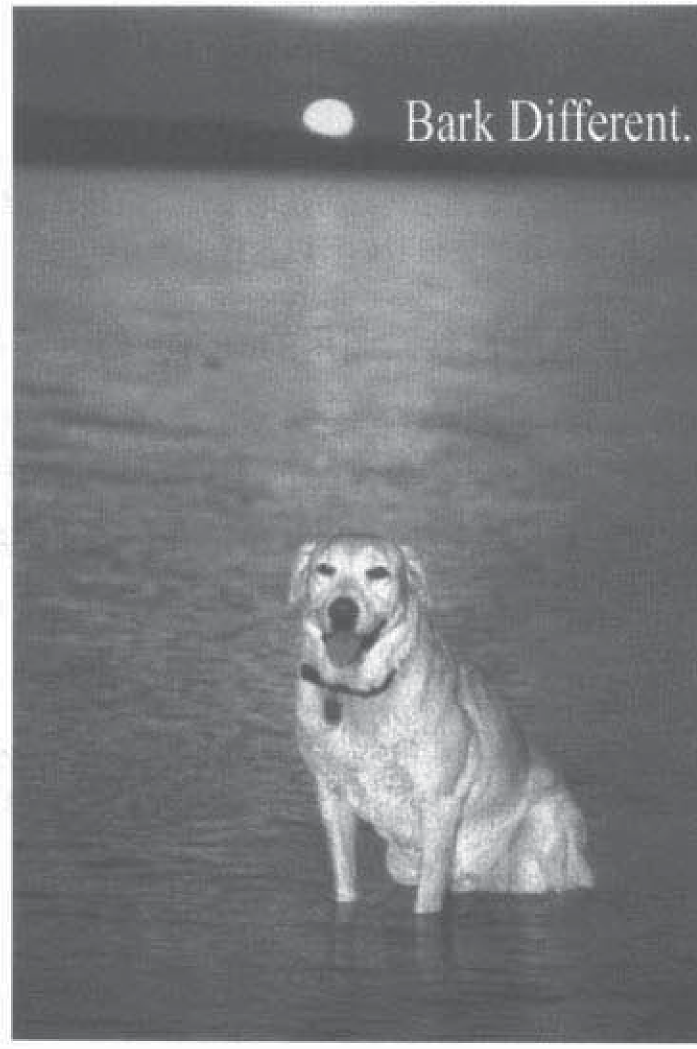
  void bark() {
    if (size > 60) {
       System.out.println("Wooof! Wooof!");
    } else if (size > 14) {
       System.out.println("Ruff!  Ruff!");
    } else {
       System.out.println("Yip! Yip!");
    }
  }
}
```

```
class DogTestDrive {

   public static void main (String[] args) {
     Dog one = new Dog();
     one.size = 70;
     Dog two = new Dog();
     two.size = 8;
     Dog three = new Dog();
     three.size = 35;

     one.bark();
     two.bark();
     three.bark();
   }
}
```

```
File Edit Window Help Playdead
%java DogTestDrive
Wooof! Wooof!
Yip! Yip!
Ruff!  Ruff!
```



你可以传值给方法

如同其他的程序设计语言，你可以传值给方法。举例来说，你可能会要告诉Dog对象叫几声：

```
d.bark(3);
```

由于不同的程序设计背景和个人喜好，你可能会用实参（argument）或形参（parameter）来调用传给方法的参数。虽然在正统学院派的信息工程领域中这两者是不同的，但我们可以这样来区分：

方法会运用形参。调用的一方会传入实参。

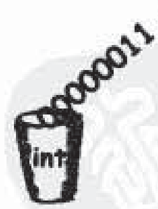
实参是传给方法的值。当它传入方法后就成了形参。参数跟局部（local）变量是一样的。它有类型与名称，可以在方法内运用。

重点是：如果某个方法需要参数，你就一定得传东西给它。那个东西得是适当类型的值。

```
Dog d = new Dog();
d.bark(3);
```

实参

1. 调用Dog上的 bark()方法，并传入“3”这个值（作为此方法的参数）。
2. 以int类型表示的值3会传递给bark()。



形参

3. 此值会传给numOfBarks这个参数（int类型的变量）。

```
void bark(int numOfBarks) {
  while (numOfBarks > 0) {
    System.out.println("ruff");
    numOfBarks = numOfBarks - 1;
  }
}
```

4. 把numOfBarks当作一般的变量使用。

你可以从方法中取返回值

方法可以有返回值。每个方法都声明返回的类型，但目前我们都是把方法设成返回 void 类型，这代表并没有返回任何东西。

```
void go() {

}
```

但我们可以声明一个方法，回传给调用方指定的类型值，如：

```
int giveSecret() {
   return 42;
}
```

如果你将一个方法声明有返回值，你就必须返回所声明类型的值！（或是与声明类型兼容的值。我们会在第7章与第8章讨论多态的时候提到更多的细节）。



编译器不会让你返回错误的类型

说好了要返回，最好就得返回！

00101010

```
int theSecret = life.giveSecret();

int giveSecret() {
    return 42;
}
```

类型必须相符

代表42的字节组合会从giveSecret()方法中返回，并指派给称为theSecret的变量

必须要与int兼容

你可以向方法中传入一个以上的参数

方法可以有多个参数。在声明的时候要用逗号分开，传入的时候也是用逗号分开。最重要的是，如果方法有参数，你一定要以正确数量、类型和顺序来传递参数。

调用需要两个参数的方法，并传入两个参数：

```
void go() {
   TestStuff t = new TestStuff();
   t.takeTwo(12, 34);
}

void takeTwo(int x, int y) {
   int z = x + y;
   System.out.println("Total is " + z);
}
```

传入的参数会以相同的顺序赋值。第一个实参会赋给第一个形参，依此类推

你也可以将变量当作参数传入，只要类型相符就可以：

```
void go() {
   int foo = 7;
   int bar = 3;
   t.takeTwo(foo, bar);
}

void takeTwo(int x, int y) {
   int z = x + y;
   System.out.println("Total is " + z);
}
```

foo与bar的值会赋给x与y，所以x的值会是7，而y的值会是3

x的值会是10

Java是通过值传递的
也就是说通过拷贝传递



```
int x = 7;
```

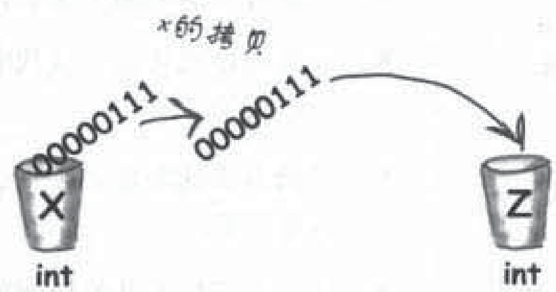


1. 声明一个int类型的变量并赋值为7。代表7的字节组合会放进称为x的变量中。

```
void go(int z){ }
```

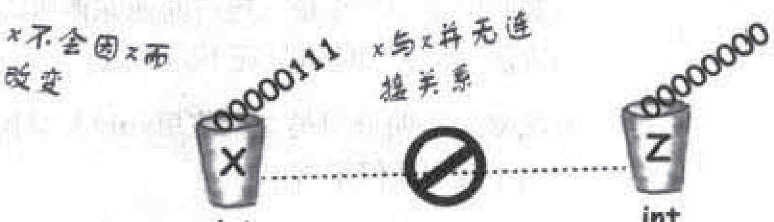


2. 声明一个有int参数的方法，参数名称为z。




```
foo.go(x);
void go(int z){ }
```

3. 以x为参数传入go()这个方法中。x的字节组合会被拷贝并装进z中。




```
void go(int z){
    z = 0;
}
```

4. 在方法中改变z的值。此时x的值不会改变！传给z的只是个拷贝。

方法无法改变调用方所传入的参数。

there are no Dumb Questions

问： 如果想要传入的参数是对象而不是primitive主数据类型会怎样？

答： 你会在稍后的章节中知道有关这件事情更多的细节，但是你早就知道答案了。在Java中所传递的所有东西都是值，但此值是变量所携带的值。还有，引用对象的变量所携带的是远程控制而不是对象本身。若你对方法传入参数，实际上传入的是远程控制的拷贝。

问： 方法可以声明多个返回值吗？有没有别的方法可以返回多个值？

答： 方法只能声明单一的返回值。若你需要返回3个int值，就把返回类型说明为int的数组，将值装进数组中来返回。如果有混合不同类型的值要返回时，等我们稍后讨论到ArrayList时再说。

问： 一定要返回所声明的类型吗？

答： 你可以返回会被隐含转换成声明类型的其他类型值。例如说用byte当作int类型的返回。但若声明的类型容器小于想要返回的类型时，必须作明确的转换。

问： 我可不可以忽略返回值？

答： Java并未要求一定要处理返回值。你可以调用返回非void类型的方法而不必理会返回值，这代表你要的是方法的行为而不是返回值。你可以不指派返回值。



提醒你：Java注重类型！

当返回类型声明成兔子的时候你不能返回长颈鹿。参数也是这样。你不能对取用兔子的参数传入长颈鹿。

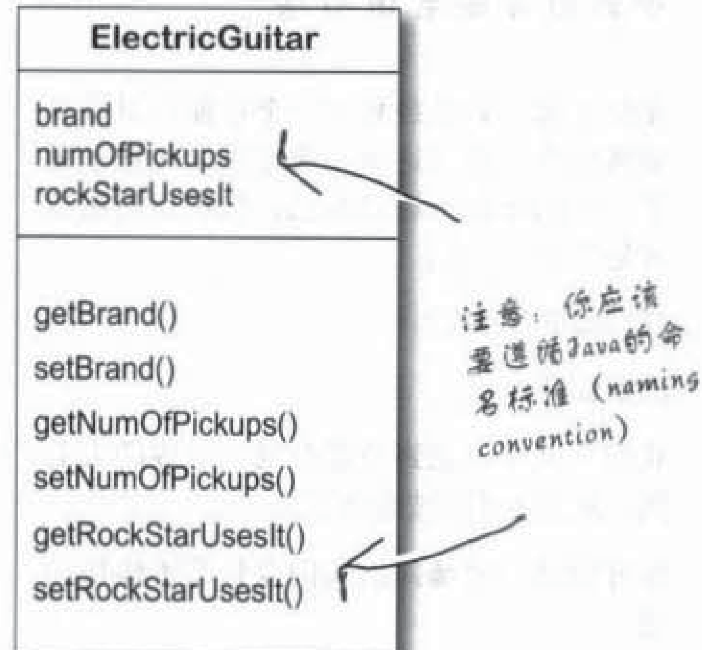
要点

- 类定义对象所知及所为。
- 对象所知者是实例变量。
- 对象所为者是方法。
- 方法可依据实例变量来展现不同的行为。
- 方法可使用参数，这代表你可以传入一个或多个值给方法。
- 传给方法的参数必须符合声明时的数量、顺序和类型。
- 传入与传出方法的值类型可以隐含地放大或是明确地缩小。
- 传给方法的参数值可以是直接指定的文字或数字（例如 2 或 ‘c’ 等）或者是与所声明参数相同类型的变量（还有其他东西可以传给方法，但我们的进度还不到那边）。
- 方法必须声明返回类型。使用void类型代表方法不返回任何东西。
- 如果方法声明了非void的返回类型，那就一定要返回与声明类型相同的值。

运用参数与返回类型

我们已经看过参数与返回类型的工作，接下来就要有效地利用了：来看Getter与Setter。如果要很正式地讨论，你会称他们为Accessor与Mutator。不过这样只是更饶舌而已，由于Getter与Getter较为符合Java的命名习惯，所以我们接下来都会这么叫它们。

Getter与Setter可让你执行get与set。Getter的目的只有一个，就是返回实例变量的值。毫无意外的，Setter的目的就是要取用一个参数来设定实例变量的值。




```
class ElectricGuitar {

   String brand;
   int numOfPickups;
   boolean rockStarUsesIt;

   String getBrand() {
      return brand;
   }

   void setBrand(String aBrand) {
      brand = aBrand;
   }

   int getNumOfPickups() {
      return numOfPickups;
   }

   void setNumOfPickups(int num) {
      numOfPickups = num;
   }

   boolean getRockStarUsesIt() {
      return rockStarUsesIt;
   }

   void setRockStarUsesIt(boolean yesOrNo) {
      rockStarUsesIt = yesOrNo;
   }
}
```



封装（Encapsulation）

不封装可能会很难堪

在此之前我们已经犯了一个在面向对象界最糟糕的错误（这可不像说被发现袜子破了一个洞这种小尴尬而已，我们说的错误可是严重的失礼）。

我们哪里有错呢？

泄露资料！

我们并没有注意到数据会被全世界的人看到，甚至还可以被改动。

你可能经历过暴露出实例变量的不愉快感觉。

暴露的意思是可通过圆点运算符来存取，像是：

```
tehCat.height = 27;
```

你可以把这件事情看做是直接通过远程控制修改Cat的实例变量。若远程控制落入不当之人的手上，变量就可能会成为杀伤力强大的武器。因为你无法防止下面的操作：

```
theCat.height = 0;
```

千万不能让这种事情发生！

这一定会很糟糕。所以我们需要创建Setter这个方法给所有的实例变量，并寻求某种方法强制其他程序都必须通过Setter来设定变量而不是直接的存取。



强迫所有人都得调用Setter，我们就可以防止Cat被设定成无法接受的高度。

```
public void setHeight(int ht) {
    if (ht > 9) {
        height = ht;
    }
}
```

这个检查可以确保高度不会低于9

数据隐藏

要将程序的实现从不良数据改成可以保护数据且让你还能修改数据的方式是很简单的。

所以要如何隐藏数据呢？答案是使用公有与私有这两个存取修饰符（access modifier）。

以下就是封装的基本原则：将你的实例变量标记为私有的，并提供公有的getter与setter来控制存取动作。或许在你有了更多的Java设计与编写经验之后会有些许不同的做法，但是目前这种做法可以维持住安全性。

将实例变量标记为private。

将getters与setters标记为public。

"老王忘记把他的猫封装，后来他的猫就被辗平了……"

（在捷运站听到的鬼故事）



Java Exposed

本周的来宾：一个即将被封装的对象引用

HeadFirst：封装有什么本事？

Object：嗯，你有没有梦到过面对500个听众时，突然发觉自己没有穿裤子？

HeadFirst：是有一次。我梦到跟一群模特儿在后宫嬉戏，然后我的裤子……呃，先不谈这个。OK，所以你是说没有封装就像没穿裤子。但是没有露一点出来会不会很不舒服？

Object：不会吧，大哥。不舒服？很不舒服？哈哈哈哈……哈哈哈……

HeadFirst：这有什么好笑的？我是很认真的。

Object：哇哈哈哈哈……（在地上滚来滚去）……哈哈哈……（泪）……哈哈哈……

HeadFirst：来人啊！叫救护车，快！

Object：我没事了……哈……啊……快不行了……好了，真的没事了……啊……（深呼吸）。

HeadFirst：好吧，请告诉我们封装可以怎样保护你的安全。

Object：封装会对我的实例变量加上绝对领域，因此没有人能够恶搞我的变量。

HeadFirst：比如说？

Object：用膝盖想也知道。大部分的实例变量值都有一个适当的范围，比如身高就不可能是负的、佛跳墙是不可能在3分钟之内做好的。

HeadFirst：我懂你的意思了。那封装是如何设下保护罩的？

Object：强迫其他的程序一定得经过setter。如此setter就能够检查参数并判断是否可以执行。setter也许可以退回不合理的值、或是抛出Exception、或者自己进行取小数点的动作。重点在于你可于setter中执行任何动作，直接暴露的public实体变量就没有这个能耐。

HeadFirst：但是我有看过某些setter什么事情也没做，只是把值设给变量而已。这样不是只会增加执行的负担吗？

Object：这对getter也是一样的，好处是你事后可以改变想法却不会需要改变其他部分的程序。假设说所有人都使用到你的类以及公有变量，万一有一天你发现这个变量需要检查，那不是所有人都要跟着改成调用setter吗？封装的优点就是能够让你三心二意却又不会伤害别人。直接存取变量的效率是比不上这个好处的。

封装 GoodDog

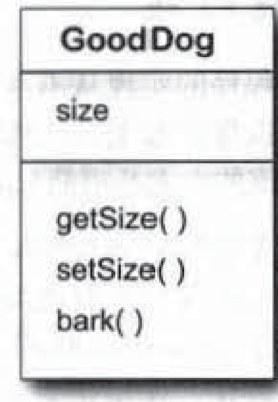
```
class GoodDog {

    private int size;

    public int getSize() {
      return size;
    }

    public void setSize(int s) {
      size = s;
    }

    void bark() {
      if (size > 60) {
         System.out.println("Wooof! Wooof!");
      } else if (size > 14) {
         System.out.println("Ruff!  Ruff!");
      } else {
         System.out.println("Yip! Yip!");
      }
    }
}
```



将实例变量设定为private的

将getter与setter设定为private的

虽然此方法没有加上实质的功能性，但最重要的是允许你能够在事后改变心意，你可以回头把程序改得更安全、更快、更好。

任何有值可以被运用到的地方，都可用调用方法的方式来取得该类型的值。

比如：

int x = 3 + 24;

可以这样改写：

int x = 3 + one.getSize();

```
class GoodDogTestDrive {

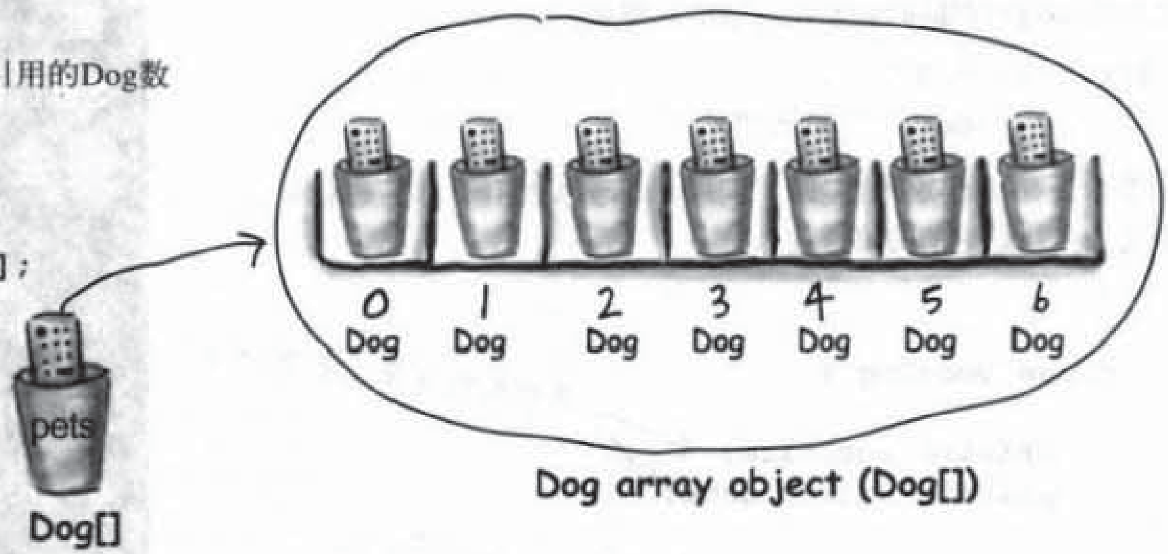
    public static void main (String[] args) {
      GoodDog one = new GoodDog();
      one.setSize(70);
      GoodDog two = new GoodDog();
      two.setSize(8);
      System.out.println("Dog one: " + one.getSize());
      System.out.println("Dog two: " + two.getSize());
      one.bark();
      two.bark();
    }
}
```

数组中对象的行为

数组中的对象就如同其他的对象一样，唯一的差别就是如何取得而已。换言之，不同处在于你如何取得遥控器。让我们先来尝试调用数组中的Dog对象。

❶ 声明一个装载7个Dog引用的Dog数组。

```
Dog[] pets;
pets = new Dog[7];
```

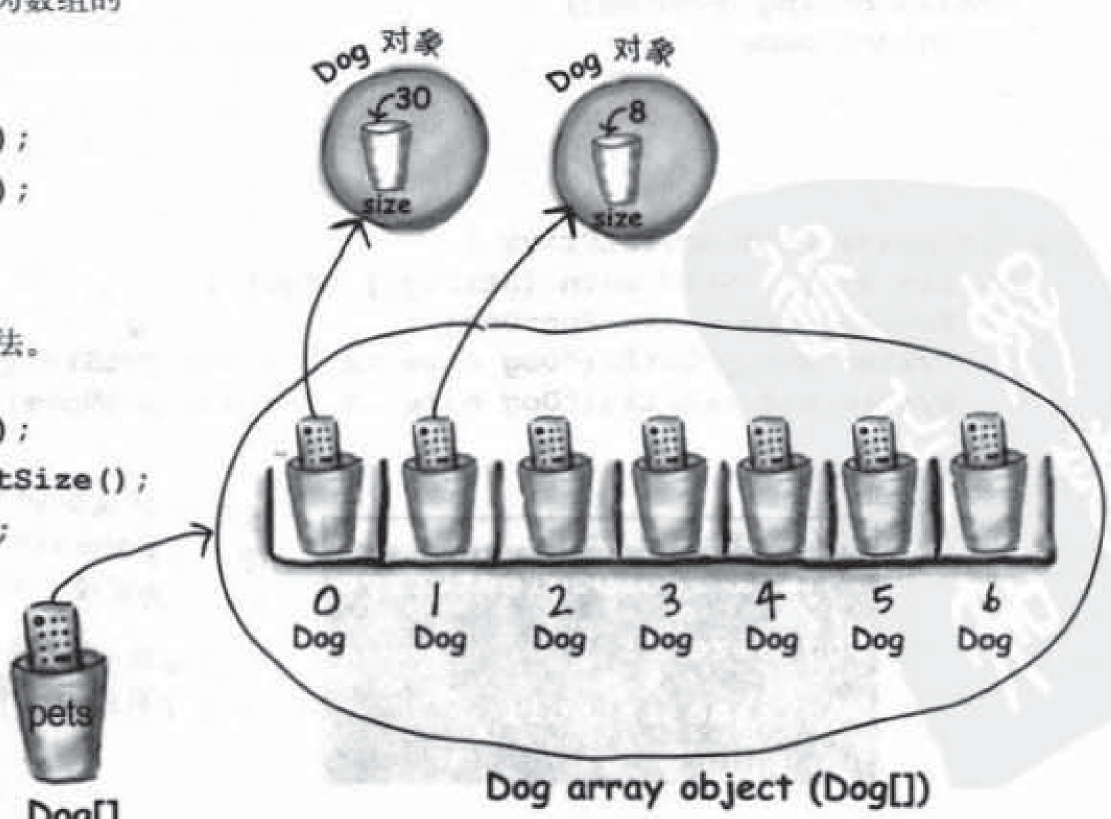


❷ 创建两个Dog对象并赋值为数组的前两项元素。

```
pets[0] = new Dog();
pets[1] = new Dog();
```

❸ 调用这两个Dog对象的方法。

```
pets[0].setSize(30);
int x = pets[0].getSize();
pets[1].setSize(8);
```



声明与初始化实例变量

你已经知道变量的声明至少需要名称与类型：

```
int size;
String name;
```

并且你也知道可以同时初始化（赋值）变量：

```
int size = 420;
String name = "Donny";
```

但如果你没有初始实例变量时，调用getter会发生什么事？也就是说实例变量在初始之前的值是什么？

实例变量永远都会有默认值。如果你没有明确的赋值给实例变量，或者没有调用setter，实例变量还是会有值！

integers	0
floating points	0.0
booleans	false
references	null

```
class PoorDog {

   private int size;
   private String name;

   public int getSize() {
      return size;
   }
   public String getName() {
      return name;
   }

}

public class PoorDogTestDrive {
   public static void main (String[] args) {
      PoorDog one = new PoorDog();
      System.out.println("Dog size is " + one.getSize());
      System.out.println("Dog name is " + one.getName());
   }
}
```

声明实例变量但是不给值

会返回什么？

你想这会通过编译吗？

```
File Edit Window Help CallVet
% java PoorDogTestDrive
Dog size is 0
Dog name is null
```

你无需初始实例变量，因为它们会有默认值。数字的primitive（包括char）的预设为0，boolean的预设为false，而对象引用则为null。

（要记得null代表没有操作对象的远程控制，它是个引用而不是对象）

实例变量与局部变量之间的差别

1. 实例变量是声明在类内而不是方法中。

```
class Horse {
    private double height = 15.2;
    private String breed;
    // more code...
}
```

2. 局部变量是声明在方法中的。

```
class AddThing {
    int a;
    int b = 12;

    public int add() {
        int total = a + b;
        return total;
    }
}
```

3. 局部变量在使用前必须初始化。

```
class Foo {
    public void go() {
        int x;
        int z = x + 3;
    }
}
```

无法编译！你可以声明没有值的x，但若要使用时编译器就会给出警示

```
File Edit Window Help Yikes
% javac Foo.java
Foo.java:4: variable x might
not have been initialized
        int z = x + 3;
1 error         ^
```

局部变量没有默认值！如果在变量被初始前就要使用的话，编译器会显示错误。

there are no Dumb Questions

问： 那方法的参数呢？局部变量的规则也适用于它们身上吗？

答： 方法的参数基本上与局部变量是相同的——它们都是在方法中声明的（精确地说应该是在方法的参数列声明的，但相较于实例变量来说它也算是局部的）。而参数并没有未声明的问题，所以编译器也不可能对这样事情显示出错误。

这是因为如果调用方法而没有赋值参数时编译器就会显示错误。所以说参数一定会被初始化，编译器会确保方法被调用时会有与声明所相符的参数，且参数会自动地被赋值进去。

变量的比较（primitive主数据类型或引用）

有时你需要知道两个primitive主数据类型是否相等。很简单，只要使用==这个运算符就可以。有时你想要知道两个引用变量是否引用到堆上的同一个对象。这也很容易，也是使用 == 运算符。但有时你会需要知道两个对象是否真的相等。此时你就得使用equals()这个方法。相等的意义要视对象的类型而定。举例来说，如果两个不同的String带有相同的字符，它们在涵义上是相等的。但对Dog来说，你认为尺寸大小或名字一样的Dog是相等的吗？所以说是否被视为相等要看对象类型而定。我们会在后面的章节继续探讨对象相等性的部分，但现在我们要知道的是==只用来比对两个变量的字节组合，实质所表示的意义则不重要。字节组合要么就是相等，要么就是不相等。

使用==来比较两个primitive主数据类型，或者判断两个引用是否引用同一个对象。

使用equals()来判断两个对象是否在意义上相等。

（像是两个String对象是否带有相同的字节组合）

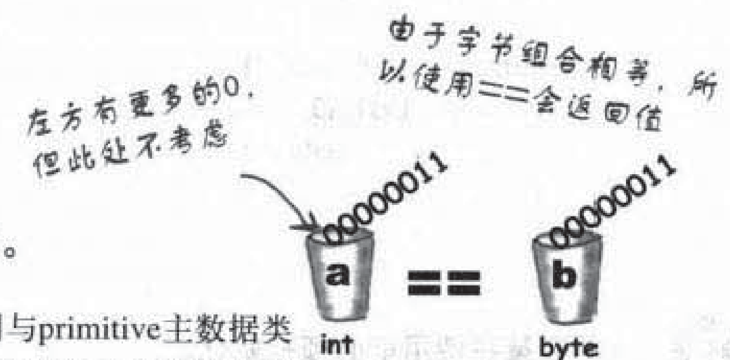
使用==来比对primitive主数据类型

这个运算式可以用来比较任何类型的两个变量，它只是比较其中的字节组合。

```
int a = 3;

byte b = 3;

if (a == b) { // true }
```



使用==来判别两个引用是否都指向同一对象。

要记得，这只是在比较字节组合的模样。此规则适用于引用与primitive主数据类型。因此==运算符对参照相同对象的引用变量会返回值。在此情况下我们还是无法得知字节组合的样式，但可以确定的是所参照的相同的对象。

```
Foo a = new Foo();

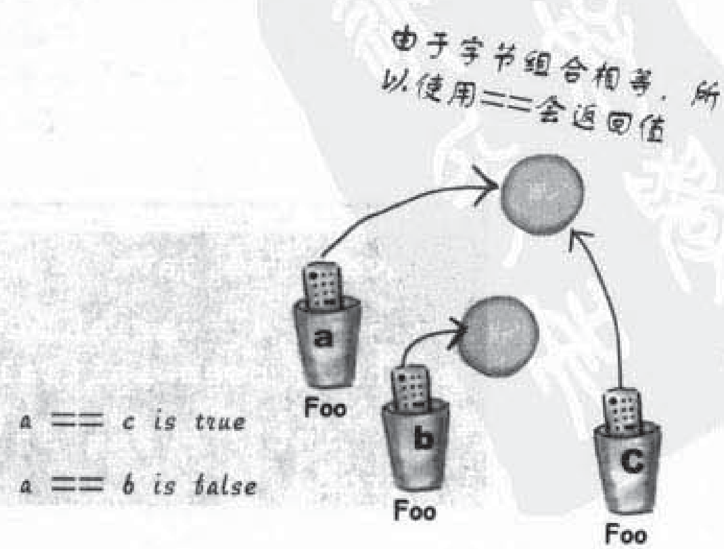
Foo b = new Foo();

Foo c = a;

if (a == b) { // false }

if (a == c) { // true }

if (b == c) { // false }
```





Make it Stick



玫瑰是红的，
有些事情是讲天分的，
以值传递就是以拷贝传递。

噢，你以为写个烂诗很容易吗？那你写个新诗来给我瞧瞧。其实只要你写个够好的诗，那你就永远不会忘记这件事。

Sharpen your pencil

哪些是合法的？



对下面这个方法来说，右边列出的哪几个调用是合法的？

在合法的述句旁边打勾。

```
int calcArea(int height, int width) {
   return height * width;
}
```

```
int a = calcArea(7, 12);
short c = 7;
calcArea(c,15);
int d = calcArea(57);
calcArea(2,3);
long t = 42;
int f = calcArea(t,17);
int g = calcArea();
calcArea();
byte h = calcArea(4,20);
int j = calcArea(2,3,5);
```



我是编译器



这一页的Java程序代码都代表一份完整的源文件。你的任务是要扮演编译器角色并判断哪个程序可以编译过关。如果有问题，哪里要修改？

A

```
class XCopy {

  public static void main(String [] args) {

    int orig = 42;

    XCopy x = new XCopy();

    int y = x.go(orig);

    System.out.println(orig + " " + y);
  }

  int go(int arg) {

    arg = arg * 2;

    return arg;
  }
}
```

B

```
class Clock {
  String time;

  void setTime(String t) {
    time = t;
  }

  void getTime() {
    return time;
  }
}

class ClockTestDrive {
  public static void main(String [] args) {

    Clock c = new Clock();

    c.setTime("1245");
    String tod = c.getTime();
    System.out.println("time: " + tod);
  }
}
```



我是谁？

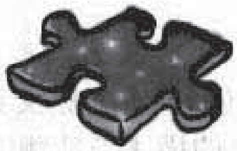


一组Java组件精心打扮出席化装舞会，中场时间有人提议要玩猜猜我是谁的游戏，你可以根据它们对自己的描述来猜测出提示的是哪位。规则是每个组件都得说实话，若某些提示同时对数个组件都为真的话，则将它们全部填入。

今晚出席舞会的有：

instance variable，argument，return，getter，setter，encapsulation，public，private，pass by value，method

一个类可以带有很多个	______
一种方法只能带有一个	______
可以被隐含地转换	______
我喜欢private的实例变量	______
其实就是制作一个拷贝	______
应该只有setter才能更新	______
方法可以带很多个	______
根据定义返回	______
不应该以实例变量来运用	______
可以有许多个参数	______
被定义成采用一个参数	______
帮忙创建封装	______
总是单飞	______



连连看

右边有一个 Java 小程序。其中有两段程序不见了。你的任务是找出下面所列出的程序段与相符的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

```
public class Mix4 {
  int counter = 0;
  public static void main(String [] args) {
    int count = 0;
    Mix4 [] m4a  =new Mix4[20];
    int x = 0;
    while (          ) {
      m4a[x] = new Mix4();
      m4a[x].counter = m4a[x].counter + 1;
      count = count + 1;
      count = count + m4a[x].maybeNew(x);
      x = x + 1;
    }
    System.out.println(count + " "
                           + m4a[1].counter);
  }

  public int maybeNew(int index) {
    if (           ) {
      Mix4 m4 = new Mix4();
      m4.counter = m4.counter + 1;
      return 1;
    }
    return 0;
  }
}
```

程序段

```
x < 9
index < 5
```

```
x < 20
index < 5
```

```
x < 7
index < 7
```

```
x < 19
index < 1
```

输出

14 7

9 5

19 1

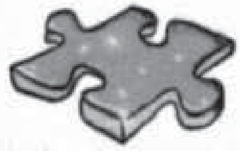
14 1

25 1

7 7

20 1

20 5



泳池迷宫



你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段不能用两次，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

输出

```
File Edit Window Help BellyFlop
%java Puzzle4
result 543345
```

```
public class Puzzle4 {
  public static void main(String [] args) {
    ___________________________

    int y = 1;
    int x = 0;
    int result = 0;
    while (x < 6) {

      ___________________

      ___________________

      y = y * 10;

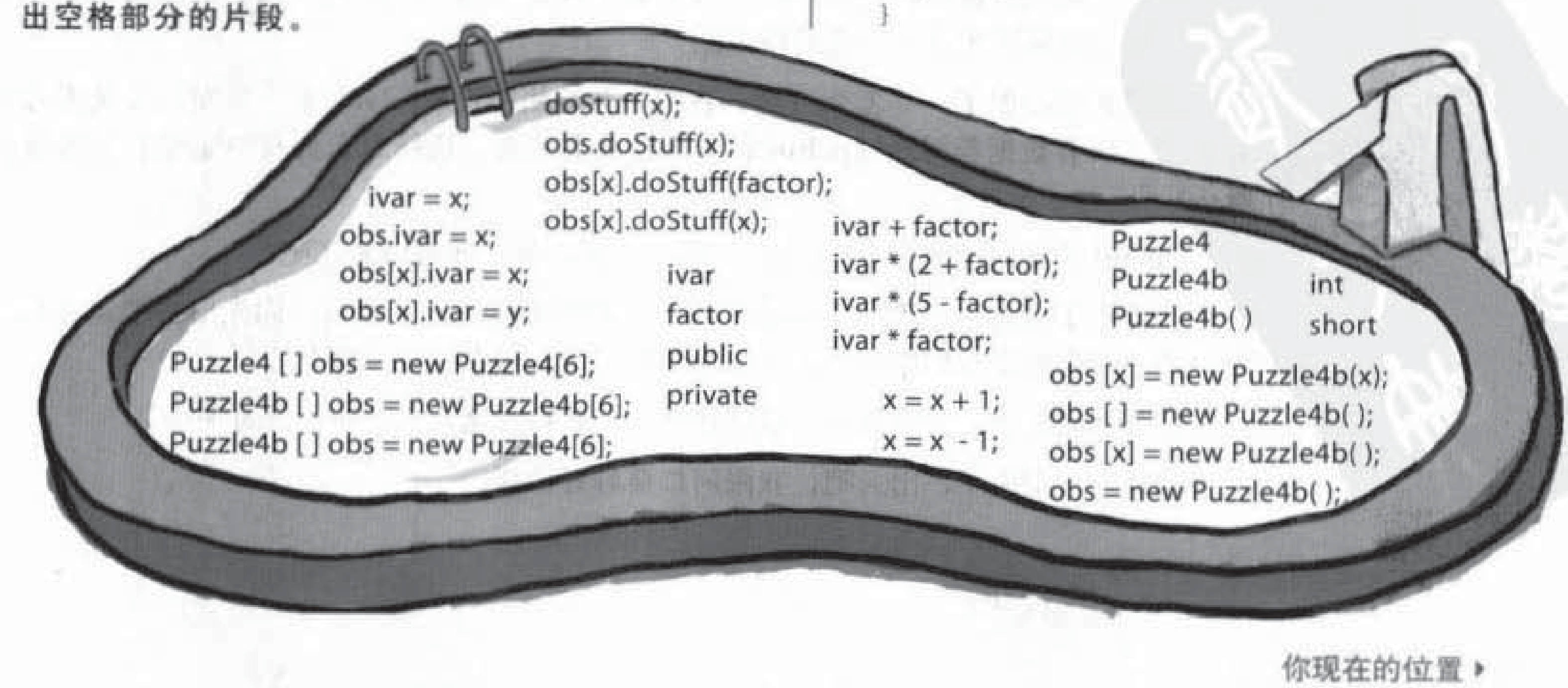
      ___________

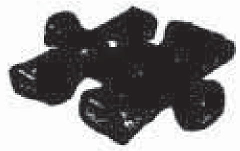
    }
    x = 6;
    while (x > 0) {

      ___________
      result = result + ___________
    }
    System.out.println("result " + result);
  }
}

class ________ {
  int ivar;
  _______ _______ doStuff(int ________) {
    if (ivar > 100) {
      return ______________
    } else {
      return ______________
    }
  }
}
```

从池中找出可以填在输出空格部分的片段。





达康之道：公私分明

当阿仁发觉有只光电鼠标指在他头上时，整个人都呆住了。他早就知道会面对今天这个状况，但万万想不到的是，拿鼠标的人居然会是傻强，更重要的是这只鼠标居然是无线的。不过这时候也没有办法去想这些事情了。傻强命令阿仁走进琛哥的办公室，琛哥早已经坐在里面等着。阿仁没有把握自己的身份是否已经曝光，这反而让他有点不知道要对琛哥说什么。

“琛哥，”阿仁决定先装傻：“我知道错了，以后我每一行程序都会加批注。”

“五年前，科学园区大门口外的达康公司开张大吉……”琛哥抽了口烟：“我和兄弟们雄心壮志，谁知道开张不到一个月，每天平均被黑客攻击1.3次，一年内挂掉6台服务器。佛祖保佑，算命的说我是一将功成万骨枯，不过我不同意……”没抽两口的烟就被捻熄了。

阿仁的身份会曝光吗？傻强到底说出了什么不该说的事？

“出来写程序的，早晚都会有漏洞”琛哥又点起了一根烟，看着傻强：“傻强，你跟我五年多了，你在这几年都很能干，现在有个问题问你，就说如果有个兄弟写程序有漏洞，你敢不敢重写？”

傻强不愧是傻的：“当然敢啊，琛哥。”

琛哥：“那你把今天发生的事情说给阿仁听。”



傻强：“漏洞是还没有找到，来攻击我们的黑客倒是抓到了，那个黑客骨头硬，我们把他抓到顶楼足足打了十分钟，十分钟都没有打错一个指令。琛哥说那个写程序的人很会掩饰，今天谁没有出现，谁就是写出漏洞的人。仁哥，我好想问你，今天追的show girl漂亮吗？因为你知道，show girl不漂亮那就没劲了。你知道，如果有个人他coding不专心又会翘班去看信息展的话，他就会写出漏洞。是你吧，仁哥？”

阿仁发现到有机会转移目标：“对不起，我是系统分析师……我看过你写的系统登录类，我觉得那里才最有可能出现漏洞……”

傻强开始心虚了：“怎么可能，我把所有方法都设成private了，外面的人是不可能存取的，所有数据都得通过public的实例变量来更新，这样怎么会有问题呢？应该不会呀……”

Bingo！阿仁差点要大叫出来：“琛哥，你看呢，我可以走了吧？”

琛哥点了点头：“等一下再走，我有事情要跟你讨论。傻强，你先出去吧，还有，这一阵子你先不要写程序……，不，你今天起就跟着泰国佬照顾仓库好了”

傻强两眼发直：“琛哥……我没错呀……我跟你这么久了……琛哥……”

琛哥：“别说了，出去吧，我跟阿仁要好好谈谈。”

练习解答

A “XCopy”编译与运行都没有问题！输出结果是 “42 84”。要记得Java是按值传递（也就是传拷贝的），所以orig这个变量的值不会被go()方法改变。

B

```
class Clock {
  String time;
  void setTime(String t) {
    time = t;
  }
  String getTime() {
    return time;
  }
}

class ClockTestDrive {
  public static void main(String [] args) {
    Clock c = new Clock();
    c.setTime("1245");
    String tod = c.getTime();
    System.out.println("time: " + tod);
  }
}
```

注意：setter 要定义出返回的类型

一个类可以带有很多个	instance variables, getter, setter,method
一种方法只能带有一个	return
可以被隐含地转换	return, argument
我喜欢private的实例变量	encapsulation
其实就是制作一个拷贝	pass by value
应该只有setter才能更新	instance variables
方不可以带很多个	argument
根据定义返回	getter
不应该以实例变量来运用	public
可以有许多个参数	method
被定义成采用一个参数	setter
帮忙创建封装	getter, setter, public, private
总是单飞	return

迷宫解答

```
public class Puzzle4 {
  public static void main(String [] args) {
    Puzzle4b [ ] obs = new Puzzle4b[6];
    int y = 1;
    int x = 0;
    int result = 0;
    while (x < 6) {
      obs[x] = new Puzzle4b( );
      obs[x] . ivar = y;
      y = y * 10;
      x = x + 1;
    }
    x = 6;
    while (x > 0) {
      x = x - 1;
      result = result + obs[x].doStuff(x);
    }
    System.out.println("result " + result);
  }
}
class Puzzle4b {
  int ivar;
  public int doStuff(int factor) {
    if (ivar > 100) {
      return ivar * factor;
    } else {
      return ivar * (5 - factor);
    }
  }
}
```

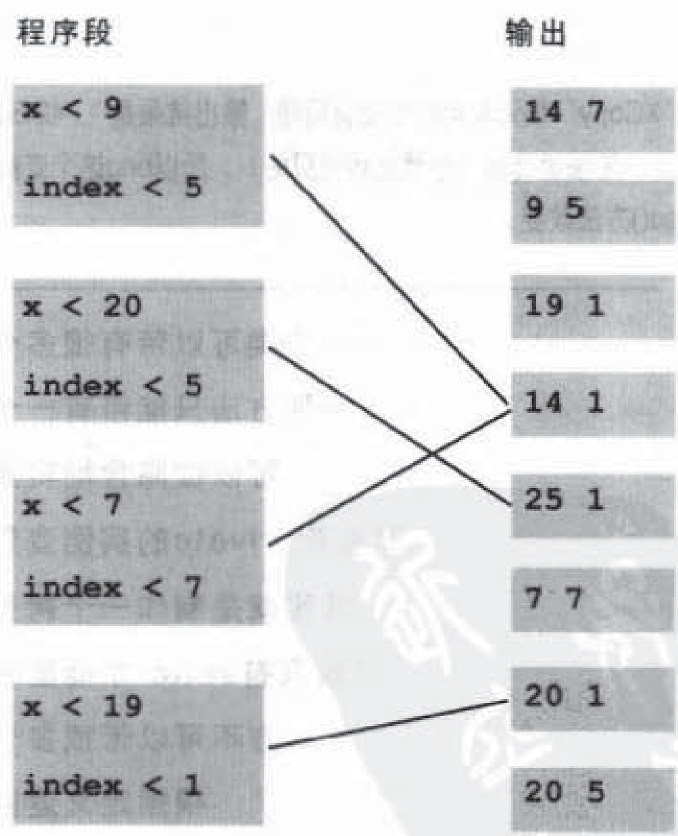
输出

```
File Edit Window Help BellyFlop
%java Puzzle4
result 543345
```

“达康之道”解析

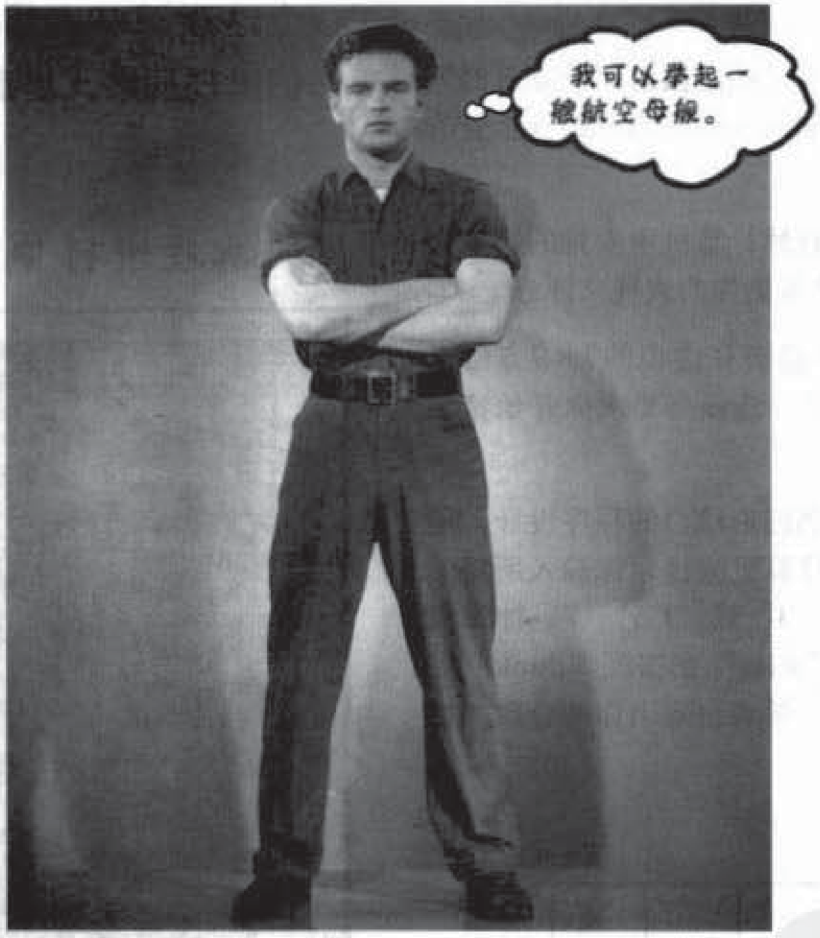
傻强犯了一个严重的错误：千万别让人知道你的概念是错的。

实例变量应该要标记为private，并通过getter与setter来存取。如此才能有机会确保实例变量值会落在合法的范围内。



5 编写程序

超强力方法



让方法产生更大的作用。 我们看过变量、操作过一些对象，并编写了一点程序，但这还不够，我们需要更多的其他操作，例如运算符等。更多的运算符才能让我们执行出比吠两声更有趣的事情。还有，我们需要的不只是while循环，for循环也是专业的配备之一。产生随机数也会很有用。将String转换成int会很酷，最好也要学起来。我们如果能真地从无到有去编写与测试一个实用的程序那会更棒。或许写个game是个好主意。那可是个不小的工程，因此得要花上两章才能完成。这一章会先创建出简单版，然后第6章再来创建出个豪华版。

创建一个类似战舰的游戏：攻击网站

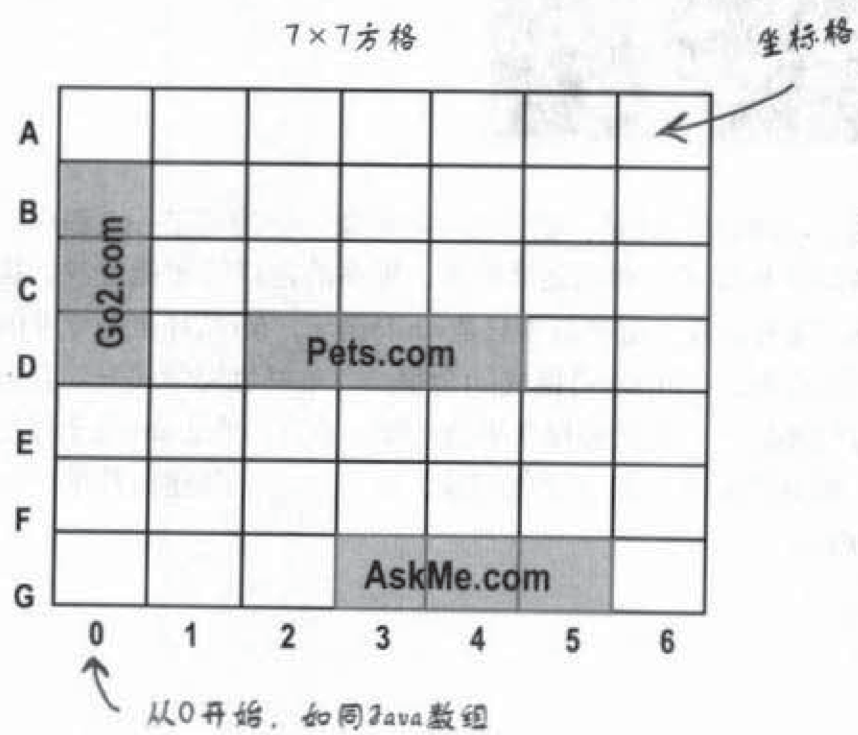
有一种棋盘类的战舰游戏，目标是要猜测对方战舰的坐标，然后轮流开炮攻击，命中数发就可以打沉对方的战舰。

不过我们不喜欢战争，只要打垮这些达康公司就好（因为与商业行为有关，如此一来本书就可以归类在经营企管的费用上）。

游戏目标：以最少的猜测次数打掉计算机所安排的达康公司（Dot Com）网站。计算机会根据你的表现来评分。

初始设置：程序启动后，计算机会在虚拟的7×7方格上安排3个达康网站。安排完成后，游戏会要求你开始猜坐标。

进行游戏：因为我们还没有学到图形接口的程序设计，所以这一版会在命令栏上进行。计算机会提示你输入所猜测的位置（格子），你会输入“A3”或“C5”等。计算机会反馈给你命中“Hit”没中“Miss”或击沉“Sunk”等回应。当你清光所有的达康时，游戏会列出你的分数并结束。



你会创建一个攻击网站游戏，它有7×7的格子与3间达康公司。每个达康网站占用3个格子。让我们重现网络大崩盘噩梦吧！

游戏进行中的画面

```
ile Edit Window Help Sell
%java DotComBust
Enter a guess  A3
miss
Enter a guess  B2
miss
Enter a guess  C4
miss
Enter a guess  D2
hit
Enter a guess  D3
hit
Enter a guess  D4
Ouch! You sunk Pets.com   : (
kill
Enter a guess  B4
miss
Enter a guess  G3
hit
Enter a guess  G4
hit
Enter a guess  G5
Ouch! You sunk AskMe.com   : (
kill
Enter a guess  A7
miss
Enter a guess  B7
miss
Enter a guess  C7
miss
Enter a guess  D7
miss
Enter a guess  E7
miss
Enter a guess  F7
miss
Enter a guess  G7
```

首先进行高层设计

我们需要类和方法，但是要哪些类和方法呢？要回答这个问题，得先要取得更多的游戏信息。

首先，我们必须了解游戏的流程。以下是基本思路：

1 玩家启动游戏。

A 计算机创建3个达康网站。

B 将此3个达康网站停在虚拟战场上。

2 游戏开始。

重复下面的操作直到所有达康网站被歼灭为止。

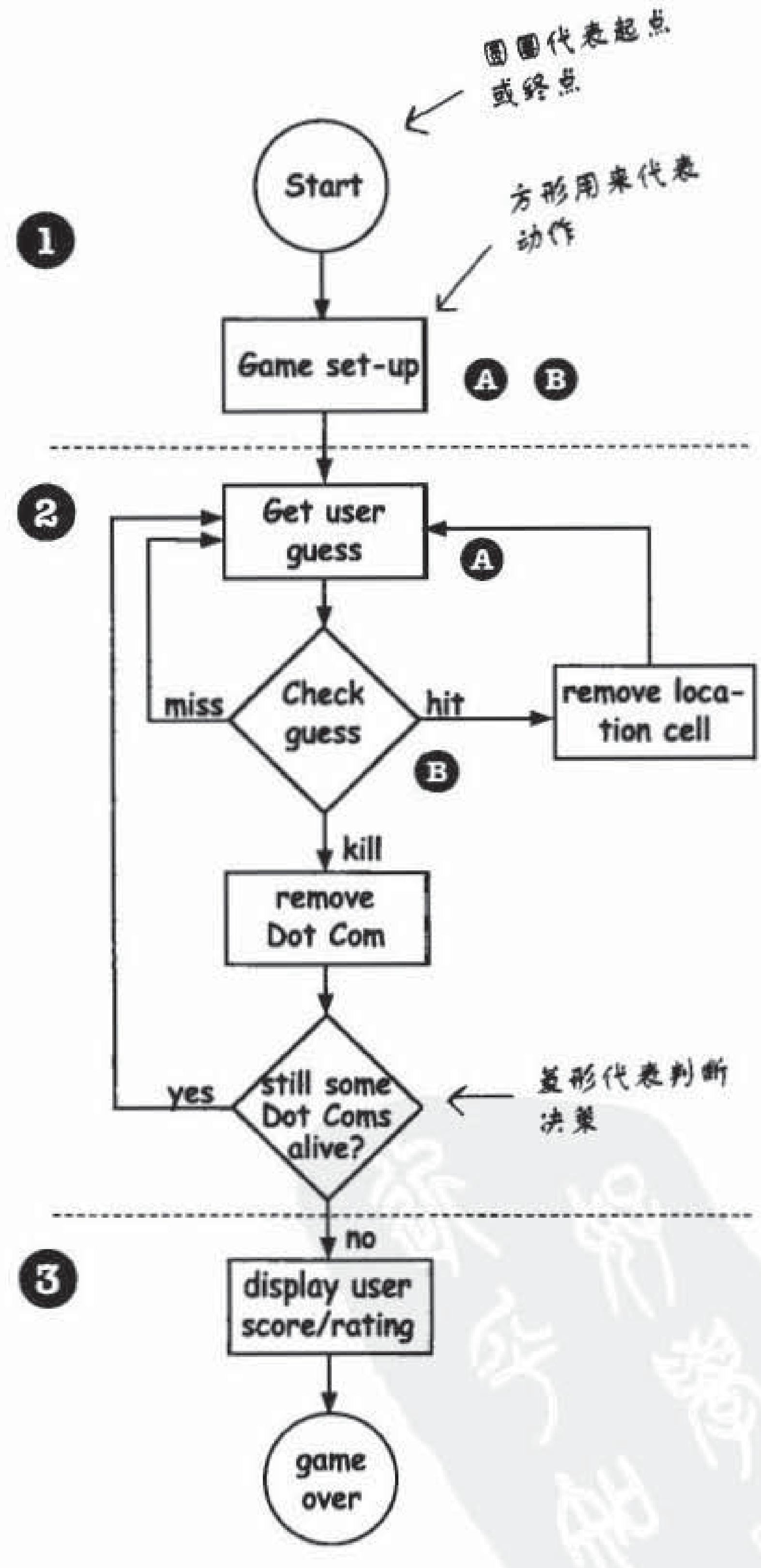
A 提示玩家输入坐标。

B 检查是否命中、没中或击沉。如果命中就删除格子，击沉就删除达康网站。

3 游戏结束。

根据猜测次数给分。

现在我们对游戏的流程有了了解。接下来的步骤是要设想出需要哪些对象。要记得以面向对象的方式来思考，专注于程序中出现的事物而不是过程。



哇！真正的流程图。

攻击网站游戏

简单的开始

看起来我们至少需要两个类：Game类和DotCom类。但在我们着手开发完整功能版之前，先从一个简单版本开始。简单版称为“Simple Dot Com Game”。这一章会创建出此版本，下一章会进行豪华版的开发工作。

该版本的东西都比较简单。相对于二维方阵，我们只使用横列，并且只设定一家达康公司。

然而游戏的目标仍然相同，因此游戏还是会需要做出DotCom的实例，将它指派在横列上，取得玩家的输入，并在所有的DotCom格子被命中时结束游戏。这个简单版能够作为豪华版的踏脚石。

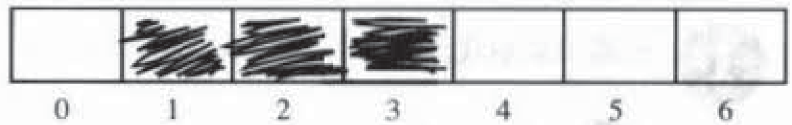
在这个简单版中，Game这个类没有实例变量，且所有的程序代码都在main()中。也就是说程序启动执行main()只会做出一个DotCom的实例，挑出一个位置来放3个连续的格子，要求玩家猜测，检查是否命中，重复这些步骤直到3格都被命中为止。



要知道虚拟的横列是虚拟的。换言之，它并没有出现在程序中，只要玩家与计算机都知道有3个连续的格子会出现在7格的横列中就好，横列并不一定要表现在程序代码中。你也许会想要用有7个int的数组来代表横列，并用其中3个元素代表达康出现的位置，但其实不需这么做。我们只需要3个元素的数组来代表DotCom占据的位置。

❶ **游戏启动。**创建单一的DotCom并指定3个格子在共7格的横列中的位置。

相对于使用“A2”这种表示法，现在的位置只需要数字就可以：



❷ **开始游戏。**提示玩家猜测，然后检查是否命中DotCom的格子。如果是，则递增numOfHits变量的值。

❸ **游戏结束。**3格都命中时游戏结束（当numOfHits的值递增到3），并告诉玩家他们花了多少次才干掉这个DotCom。

游戏画面

File Edit Window Help Destroy

```
%java SimpleDotComGame
enter a number  2
hit
enter a number  3
hit
enter a number  4
miss
enter a number  1
kill
You took 4 guesses
```

开发类

身为一个程序员，你或许会有编写程序的方法论（methodology）/过程（process）/步骤（approach）。嗯，我们也有。我们的顺序设计是先通过编写程序来帮助你了解和学习我们的想法，并不需要遵循我们实际上怎样写程序的方式。当然啦，实际工作时，你会遵循个人、项目或客户的规范。但我们完全是依照我们自己的想法去做的。当我们在创建Java的类以当作“学习经验”时，程序会像下面这样：

- ❑ **找出类应该做的事情。**
- ❑ **列出实例变量和方法。**
- ❑ **编写方法的伪码（稍后说明）。**
- ❑ **编写方法的测试用程序。**
- ❑ **实现类。**
- ❑ **测试方法。**
- ❑ **除错或重新设计。**
- ❑ **邀请辣妹参加庆功派对（没有成功过）**

我们会为每个类写出下列的事物：

伪码	测试码	真实码

这些标记会用在接下来的几页以显示出我们正在讨论哪个部分。举例来说，下面的标记表示我们正在处理SimpleDotCom类的伪码。

SimpleDotCom 类

伪码	测试码	真实码

伪码

伪码能帮你专注于逻辑而不需要顾虑到程序语法。

测试码

测试用的程序代码。

真实码

实际设计出的真正Java程序代码。

BRAIN POWER

开始编写程序时，你要决定哪个类先创建出来呢？假设某些类需要同时运用到多个类（如果你遵循良好的面向对象原则，并且没有让单一的类执行太多的任务），又该从哪里开始呢？



伪码 | 测试码 | 真实码

SimpleDotCom

int [] locationCells
int numOfHits

String checkYourself(String guess)
void setLocationCells(int[] loc)

看过下面的范例你就会对伪码如何运行有个基本的了解。伪码是介于真正的Java程序与正常英语之间的一种语言。伪码大致上包括3部分：实例变量的声明、方法的声明和方法的逻辑。伪码最重要的部分是方法的逻辑，因为它定义出会发生“什么事”，这个部分会在稍后真正编写程序代码时转译成“如何”发生。

DECLARE an int array to hold the location cells. Call it locationCells.

DECLARE an int to hold the number of hits. Call it numOfHits and SET it to 0.

DECLARE a checkYourself() method that takes a String for the user's guess ("1", "3", etc.), checks it, and returns a result representing a "hit", "miss", or "kill".

DECLARE a setLocationCells() setter method that takes an int array (which has the three cell locations as ints (2,3,4, etc.).

```
METHOD: String checkYourself(String userGuess)
    GET the user guess as a String parameter
    CONVERT the user guess to an int
    REPEAT with each of the location cells in the int array
        // COMPARE the user guess to the location cell
        IF the user guess matches
            INCREMENT the number of hits
            // FIND OUT if it was the last location cell:
            IF number of hits is 3, RETURN "kill" as the result
            ELSE it was not a kill, so RETURN"hit"
            END IF
        ELSE the user guess did not match, so RETURN "miss"
        END IF
    END REPEAT
END METHOD
```

```
METHOD: void setLocationCells(int[] cellLocations)
    GET the cell locations as an int array parameter
    ASSIGN the cell locations parameter to the cell locations instance variable
END METHOD
```

伪码 测试码 真实码

编写方法的实现部分

开始编写真正可用的方法程序代码

在我们开始编写方法之前，让我们先退回一步来写出测试方法用的程序代码。没错，我们会在有东西可以测试前就先写出测试用的部分！

先编写测试用程序代码的概念来自于极限编程（XP）方法论，这样做会让你能够更容易与更快地写出程序代码。我们并不强制采用极限编程XP，但觉得这个概念真的很不错，并且极限编程XP听起来也很酷。



极限编程（XP）

极限编程（XP）是一种新型的软件开发方法论。它的构想是结合了许多种“程序员真想这么做”的方法而成的。XP的概念于20世纪90年代出现，并已经被从两人工作室到福特汽车等级的大企业所采用。XP的推进力来自于客户会得到他想要的、想要的时候就能够取得甚至在开发过程后期变更规格时也是如此。

XP是由一组被证明有效的施行方法所组成的，这些方法都是被设计来共同运作，但许多人只选择性地实行部分的XP规则。这些方法包括了：

(1) 多次经常性的小规模发布。

(2) 避免加入规格没有的功能（不管“未来”会用到的功能性有多诱人）。

(3) 先写测试用的程序。

(4) 正常工作上下班。

(5) 随时随地重构（refactor），也就是改善程序代码。

(6) 保持简单。

(7) 双双结伴进行工作，并经常交换伴侣（不是说那个）以便让大家都清楚全局。

建议阅读专门的书籍，以免一知半解地胡乱应用。

伪码　测试码　真实码

为SimpleDotCom编写测试码

我们需要写出能够创建SimpleDotCom对象并加以测试的程序代码。对SimpleDotCom这个类来说，我们真正在意的只有checkYourselft()方法，然而我们还要实现出setLocationCells()方法以便让 checkYourself( ) 方法能够正确地执行。

先来看下面checkYourself()方法这个方法的伪码（setLocationCells()是个用手肘想也知道的setter，所以我们不用花太多时间去关心，但真正的应用程序会需要更稳固的setter，此时就会需要加以测试）。

然后自问：如果checkYourself()方法已经写好的话，我要用什么样的测试码才能证明这个方法能够正确地运行？

下面是伪码：

```
METHOD String checkYourself(String userGuess)
GET the user guess as a String parameter
CONVERT the user guess to an int
REPEAT with each of the location cells in the int array
   // COMPARE the user guess to the location cell
   IF the user guess matches
      INCREMENT the number of hits
      // FIND OUT if it was the last location cell:
      IF number of hits is 3, RETURN "Kill" as the result
      ELSE  it was not a kill, so RETURN"Hit"
      END IF
   ELSE the user guess did not match, so RETURN "Miss"
   END IF
 END REPEAT
END METHOD
```

应该要测试的部分：

（1）SimpleDotCom对象的初始化。

（2）赋值位置（带有3个int的数组，像是 {2, 3, 4}）。

（3）创建代表玩家猜设的字符串（“2”或“0”等）。

（4）传入伪造的玩家猜测来叫用checkYourself()方法。

（5）列出结果以观察是否正确。

伪码 测试码 真实码

SimpleDotCom的测试码

```
public class SimpleDotComTestDrive {

   public static void main (String[] args) {

      SimpleDotCom dot = new SimpleDotCom();

      int[] locations = {2,3,4};

      dot.setLocationCells(locations);

      String userGuess = "2";

      String result = dot.checkYourself(userGuess);

      String testResult = "failed";

      if (result.equals("hit") ) {

          testResult = "passed";

       }

       System.out.println(testResult);

   }

}
```

初始化一个SimpleDotCom对象

创建带有dot com位置的数组

调用dot com的setter

假的猜测

调用被测方法并传入假的数据

测试应该要返回"hit"才算成功

列出测试结果

there are no Dumb Questions

问： 可能是我自己没搞清楚，但是你要如何测试还没有出现的东西呢？

答： 我们从未说过要开始进行测试。编写测试用程序代码的同时确实没有东西可测，因此在写出"stub"程序代码前你应该无法编译。

问： 那我还是不懂，为什么不先写好程序然后再制作测试用的代码？

答： 思索与编写测试用的程序代码能够帮助你了解被测的程序应该要做哪些事情。

当你的实作程序代码完成时，你就能够有准备好地测试代码来进行验证。此外，你心里也有数，若没有先把它做出来，那么以后也不会去做。

理想上，先写出一点点的测试码，然后只编写能够通过该测试的方法就好。之后再编写另一点测试码，再编写新的实现以让测试能够通过。经过如此的循环，你就能够证明新加入的程序代码不会破坏原有已经测试过的部分。

Sharpen your pencil

接下来会继续实现SimpleDotCom这个类，当然也会回头修改这个测试用的类。你认为我们还需要加入什么？我们还缺了哪些测试？请写下你的看法。

伪码 | 测试码 | 真实码

checkYourself()方法

从伪码到真正的Java程序代码之间并没有完美的对应，你会看到一些调整。伪码让我们对于程序代码需要做什么有比较好的概念，接下来我们就可以看出来要如何做。

观察此程序代码的同时，要想想看如何将它改善。有标记 ① 符号的部分是我们还没有讨论过的语法与功能。后面的章节内容会加以说明。

取得玩家的猜测
```
public String checkYourself(String stringGuess) {
```
把用户猜测转换成 int
```
    int guess = Integer.parseInt(stringGuess);   ① ← 把字符串转换成int

    String result = "miss";   ← 创建出保存返回结果的变量。以miss作为默认值
```
对每个格子重复
```
    for (int cell : locationCells) {   ② ← 以循环对每个格子重复执行
```
如果猜中
```
        if (guess == cell) {   ← 比较格子与猜测值

            result = "hit";   ← 命中！
```
递增命中数
```
            numOfHits++;   ③

            break;   ④ ← 已经离开循环，但需要判断是否击沉

        } // end if

    } // end for
```
如果命中数为3
```
    if (numOfHits == locationCells.length) {
```
返回击沉信息
```
        result = "kill";   ← 已经离开循环，但需要判断是否击沉

    } // end if
```
列出信息
```
    System.out.println(result);   ← 将结果显示出来

    return result;   ← 将结果返回给调用方

} // end method
```

伪码 测试码 真实码

新功能

这一页说明之前没有讨论过的功能。其他的细节在后面还会继续探讨。这足够让你继续下去。

1. 将string转换成 int

Java内建类 / Integer的一个方法，能够将String解析 / 采用String参数

```
Integer.parseInt("3")
```

2. for循环

循环的声明：对locationCells中每个元素执行一次，每次循环都会将内容赋给cell变量

可以把（:）符号读作in，也就是for each int in location Cells...

```
for (int cell : locationCells) { }
```

声明出带有数组元素的变量。在循环的每次循环中，此变量的值都会带有不同的数组元素，直到跳出循环为止。

要被逐个执行过的数组，每循环一次数组的下一个元素都赋给变量"cell"

3. 后递增（post-increment）操作符

++ 代表将其内值加1

```
numOfHits++
```

这与 numOfHits=numOfHits+1的意思相同

4. 中止指令

```
break;
```

无条件立即跳出循环

伪码 | 测试码 | **真实码**

SimpleDotCom与SimpleDotComTester的最终版本

```
public class SimpleDotComTestDrive {

   public static void main (String[] args) {
      SimpleDotCom dot = new SimpleDotCom();
      int[] locations = {2,3,4};
      dot.setLocationCells(locations);
      String userGuess = "2";
      String result = dot.checkYourself(userGuess);
   }
}
```

```
public class SimpleDotCom {

   int[] locationCells;
   int numOfHits = 0;

   public void setLocationCells(int[] locs) {
      locationCells = locs;
   }

   public String checkYourself(String stringGuess) {
      int guess = Integer.parseInt(stringGuess);
      String result = "miss";
      for (int cell : locationCells) {
         if (guess == cell) {
            result = "hit";
            numOfHits++;
            break;
         }
      } // out of the loop

      if (numOfHits ==
           locationCells.length) {
           result = "kill";
      }
      System.out.println(result);
         return result;
   } // close method
} // close class
```

虽然能够编译与运行，但程序其实有些问题，我们稍后会讨论。

执行此程序会有什么结果？

测试用程序代码会创建SimpleDotCom对象并将位置指定为 2,3,4。然后它会以2来伪造猜测传给checkYourself()方法。

若正确执行会有下面这样的输出：

```
java SimpleDotComTestDrive
hit
```

there are no Dumb Questions

问： **如果传给parseInt()的不是数字会怎样？**

答： 这个方法只会对代表数字的String产生作用。除此之外，程序就会崩溃（崩溃的意思是抛出一个异常，后面会有讨论异常的章节）。

问： **本书前面开头的地方有个for循环的范例与此处的范例大大的不同——是否有两种不同风格的for循环？**

答： 是的！Java一开始有下面这种 for 循环：

```
for(int i=0; i < 10; i++) {
     //do something
}
```

但从Java 5.0开始你可以对数组（或其他集合）使用加强版for循环。当然你还是可以对数组使用原始风格的for循环，只是使用加强版会比较好写。

伪码 测试码 真实码

Sharpen your pencil

我们创建出SimpleDoCom类和测试类。但我们确实还没有把游戏作出来。上一页的程序代码以游戏规格为准，写出你的游戏伪码。我们列出了几行当作提示。实际的游戏程序代码在下一页，因此你要作答后才能翻到下一页。

答案应该介于12～18行之间。

METHOD public static void main (String [] args)

DECLARE an int variable to hold the number of user guesses, named numOfGuesses

COMPUTE a random number between 0 and 4 that will be the starting location cell position

WHILE the dot com is still alive :

GET user input from the command line

SimpleDotComGame
需要有下面的功能：

1. 创建出SimpleDotCom对象。
2. 赋值给它。
3. 要求玩家猜测。
4. 检查猜测值。
5. 重复猜测直到击沉为止。
6. 显示玩家的猜测次数。

游戏进行画面：

```
File Edit Window Help Runaway
%java SimpleDotComGame
enter a number  2
hit
enter a number  3
hit
enter a number  4
miss
enter a number  1
kill
You took 4 guesses
```

伪码 测试码 真实码

SimpleDotComGame类的伪码

所有事情都发生在main( )中

有些事情必须要依靠信仰。举例来说，有一行伪码说要“从命令行取得输入”。其实我们不会想要自己从无到有地完成这项功能。幸好我们使用的是面向对象。这代表你可以要求其他已经写好的类或对象来完成这件事而不必考虑它是怎么达成的。当你在编写伪码时，应该要假设你总有办法可以做出某些功能，如此才能让你专注于逻辑设计。

```
public static void main (String [] args)
    DECLARE an int variable to hold the number of user guesses, named numOfGuesses, set it to 0.
    MAKE a new SimpleDotCom instance
    COMPUTE a random number between 0 and 4 that will be the starting location cell position
    MAKE an int array with 3 ints using the randomly-generated number, that number incremented by 1,
    and that number incremented by 2 (example: 3,4,5)
    INVOKE the setLocationCells() method on the SimpleDotCom instance
    DECLARE a boolean variable representing the state of the game, named isAlive. SET it to true

    WHILE the dot com is still alive (isAlive == true) :
        GET user input from the command line
        // 检查用户的猜测
        INVOKE the checkYourself() method on the SimpleDotCom instance
        INCREMENT numOfGuesses variable
        // 判断是否击沉
        IF result is "kill"
            SET isAlive to false (which means we won't enter the loop again)
            PRINT the number of user guesses
        END IF
    END WHILE
END METHOD
```

学习技巧



每次使用单边大脑的时间不要太久。连续使用左边大脑30分钟就如同连续使用左臂30分钟一样。周期性地交换以让大脑两侧能够轮流休息。左脑活动包括了循序渐进的工作、解决逻辑问题与分析，而右脑活动包括了隐喻、创造性思考、模式匹配与可视化。

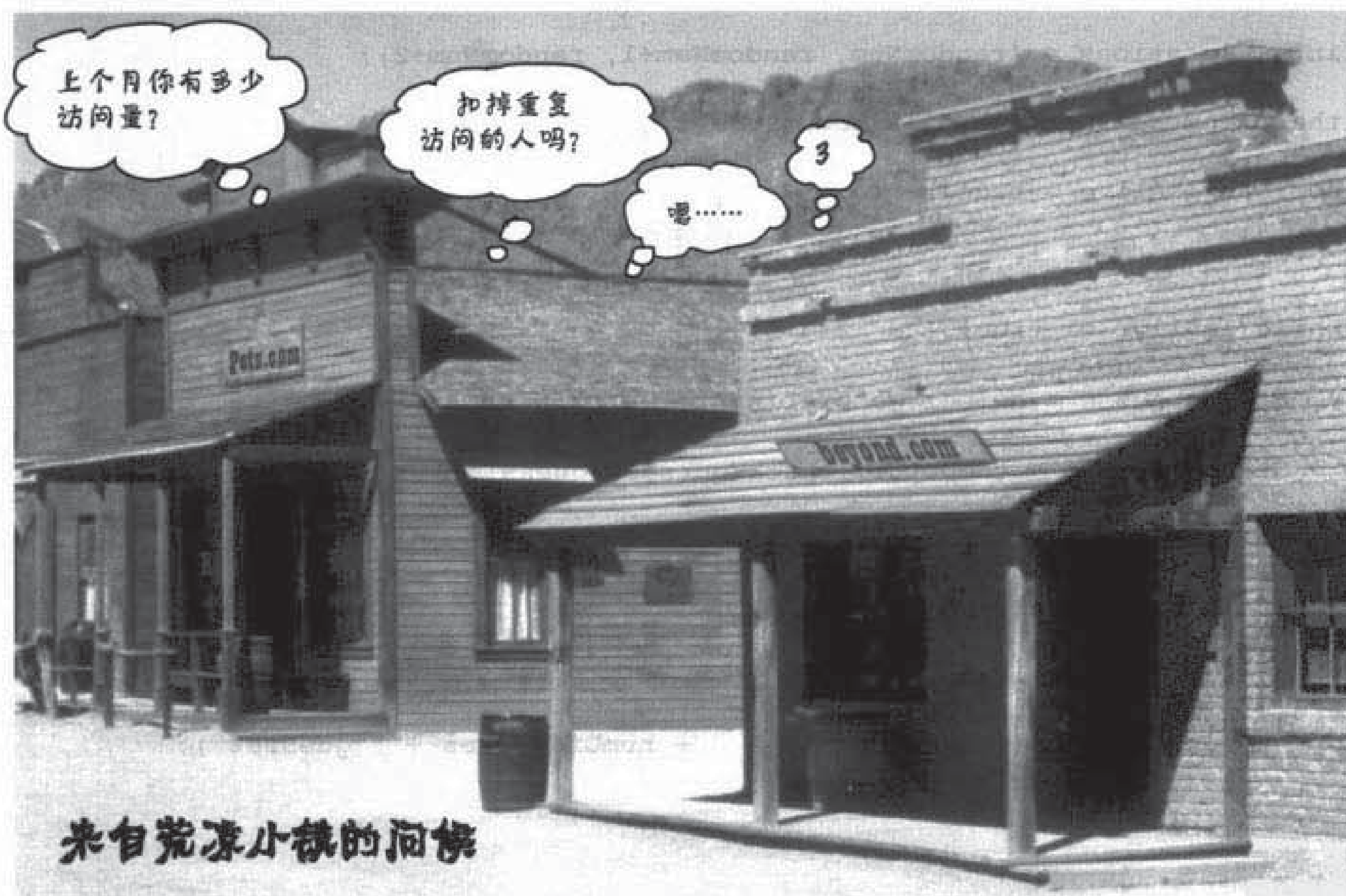
要点

- 你的Java程序应该从高层的设计开始。
- 你通常会在创建新的类时写出下列3种东西：

  伪码

  测试码

  真实码
- 伪码应该描述要做什么事情而不是如何做。
- 使用伪码来帮助测试码的设计。
- 实现方法之前应该要编写测试码。
- 如果知道要执行多少次，应该要使用for循环而不是while循环。
- 使用前置或后置的递增为变量加1（比如x++）。
- 使用前置或后置的递减来对变量减1（比如x--）。
- 使用Interger.parseInt()来取得String的整数值。
- Interger.parseInt()只会在所给的String为数字时有作用。
- 使用break命令来提前跳出循环。



伪码 | 测试码 | **真实码**

游戏的main()方法

如同你对SimpleDotCom这个类所做的一样，也要想到如何能够改进这部分的程序代码。有标记数字符号 ① 的地方是我们会加以讨论的部分。下一页会加以说明。你可能会怀疑我们为何跳过这个类的测试程序，事实上我们不需要此游戏的测试程序。它只有一个方法，所以还有需要另外作一个类来调用这个main()吗？

```
public static void main(String[] args) {

   int numOfGuesses = 0;

   GameHelper helper = new GameHelper();

   SimpleDotCom theDotCom = new SimpleDotCom();
                                  ①
   int randomNum = (int) (Math.random() * 5);

   int[] locations = {randomNum, randomNum+1, randomNum+2};

   theDotCom.setLocationCells(locations);

   boolean isAlive = true;

   while(isAlive == true) {
                               ②
      String guess = helper.getUserInput("enter a number");

      String result = theDotCom.checkYourself(guess);

      numOfGuesses++;

      if (result.equals("kill")) {

         isAlive = false;

         System.out.println("You took " + numOfGuesses + " guesses");

      } // close if

   } // close while

} // close main
```

- int numOfGuesses = 0; ← 记录玩家猜测次数的变量
- GameHelper helper = new GameHelper(); ← 我们会写出这个类来取得玩家的输入，现在先假装这是Java提供的
- SimpleDotCom theDotCom = new SimpleDotCom(); ← 创建dot com对象
- int randomNum = (int) (Math.random() * 5); / int[] locations = ... ← 用随机数产生第一格的位置，然后以此制作出数组
- theDotCom.setLocationCells(locations); ← 赋值位置
- boolean isAlive = true; / while(isAlive == true) ← 创建出记录游戏是否继续进行的boolean变量，这会用在while循环中
- String guess = ... ← 取得玩家输入的字符串
- String result = ... ← 检查玩家的猜测并将结果存储在String中
- numOfGuesses++; ← increment guess count
- if (result.equals("kill")) / isAlive = false; ← 是否击沉？若击沉，则设定isAlive为false并印出猜测次数

伪码 | 测试码 | 真实码

random()与getUserInput()

有两个部分需要加以说明。这只是个帮助你继续进行下去的解释，在本章的后面有更多关于GameHelper的详细讨论。

① 产生随机数

这是一种"类型转换(cast)"，它会强制编译器以所设定的类型来进行处理。因为Math.random会返回double类型，而我们需要0～4之间的int，因此需要将它转换成int

Math.random会返回一个介于0到小于1之间的数，所以这个公式会产生出介于0～4之间的整数

```
int randomNum = (int) (Math.random() * 5)
```

声明一个保存随机数的变量

Java内建的类

Math这个类内含的一个方法

② 取得玩家输入

这是辅助性类的一个实例，它来自于我们还没有讨论到的GameHelper

这个方法会以String参数当作提示来要求玩家输入一个猜测的数字

```
String guess = helper.getUserInput("enter a number");
```

我们声明出这个变量来保存玩家输入的猜测值

GameHelper的一个方法，它会在印出提示字符串后等待玩家输入

伪码 | 测试码 | 真实码

最后一个类：GameHelper

dot com的类已经写好了。

game的类也是。

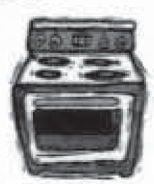
还剩下helper的类——带有getUserInput()方法的这个。此程序代码会从命令行取得输入。其中有许多细节已经超出这一章所要讨论的范围，因此我们会把这些部分留在后面（第14章）再加以说明。

只要将下列的程序代码拷贝*出来编译成名称为GameHelper的类就可以。将SimpleDotCom、SimpleDotComGame和GameHelper这3个类放到同一个目录中。

只要看到这个标记 

就代表把这个程序代码拿来用就对了。你可以认为它是没有问题的。我们稍后会讨论到它是如何运行的。






```
import java.io.*;
public class GameHelper {
  public String getUserInput(String prompt) {
     String inputLine = null;
     System.out.print(prompt + "  ");
     try {
      BufferedReader is = new BufferedReader(
      new InputStreamReader(System.in));
      inputLine = is.readLine();
      if (inputLine.length() == 0 )  return null;
    } catch (IOException e) {
      System.out.println("IOException: " + e);
    }
    return inputLine;
  }
}
```

*我们知道你很喜欢打字，但是如果你没时间的话，可以到wickedlysmart.com来下载已经打好的Ready-bake 程序代码。

玩游戏

下面就是执行游戏，输入1，2，3，4，5，6的结果，看起来还不错！

完整的程序交互
(你的画面可能会不同)

```
File Edit Window Help Smile
%java SimpleDotComGame
enter a number  1
miss
enter a number  2
miss
enter a number  3
miss
enter a number  4
hit
enter a number  5
hit
enter a number  6
kill
You took 6 guesses
```

这是什么？bug吗？

下面就是运行游戏，输入1，1，1 的结果。

```
File Edit Window Help Faint
%java SimpleDotComGame
enter a number  1
hit
enter a number  1
hit
enter a number  1
kill
You took 3 guesses
```

Sharpen your pencil



危机！

我们能找出bug吗？

我们能修改bug吗？

我们会在下一章回答这些问题 以及更多的信息……你能看出哪里有问题并加以修复吗？

关于for循环

我们已经讨论过本章的游戏程序代码（下一章会继续开发豪华版）。之前还不打算用细节和背景信息等杂事来扰乱你，所以这些东西现在才来讨论。我们会从for循环开始，但如果你是C++程序设计高手，那么可以跳过这几页。

基本（非加强版）的for循环

```
for(int i = 0; i < 100; i++){ }
```

初始化（int i = 0）

boolean测试（i < 100）

重复表达式（i++）

后递增运算符（++）

重复执行的部分放在这里（{ }）



上面这行程序以中文来说表示：“重复100次”

而编译器会这么认为：

（1）创建变量i并赋值为0。

（2）只要i小于100就重复执行。

（3）在每趟重复过程最后把 i加1。

第一段：初始化。

使用这个部分来声明和初始化用在循环体内的变量。你通常会将此变量作为计数器。实际上你可以在这里初始一个以上的变量，但我们稍后才会讨论。

第二段：boolean测试。

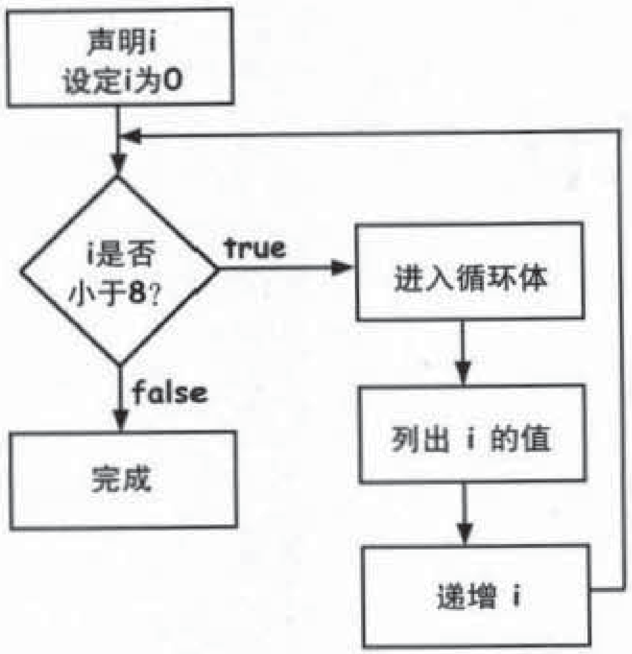
测试条件摆在这里。不管写了什么，这里一定要算出一个boolean值（true或false）。你可以安置x>－4这种测试，或者可以调用会返回boolean值的方法。

第三段：重复表达式。

在这里安置每趟循环运行完后要执行的项目。要记得这会在运行完一圈之后才会执行。

循环之旅

```
for (int i = 0; i < 8; i++) {
   System.out.println(i);
}
System.out.println("done");
```



输出：

```
File Edit Window Help Repeat
%java Test
0
1
2
3
4
5
6
7
done
```

比较for循环与while循环的差别

while循环只有boolean测试，它并没有内建的初始化或重复表达式。while适合用在不知道要循环几次的循环上。若你知道要执行几次，则使用for会比较容易阅读。下面是使用while的相同循环：

```
int i = 0;          ← 必须得声明并
                      初始化计数器
while (i < 8) {       变量
   System.out.println(i);
   i++;             ← 将计数器递增
}
System.out.println("done");
```

++ --

前置与后置的递增/递减操作符

这是加1或减1的快捷方式。

```
x++;
```

这一行与下面有相同的功能：

```
x = x + 1;
```

它们都代表：

"将x的目前值加一后存放在x中"

```
x--;
```

这一行与下面有相同的功能：

```
x = x - 1;
```

加入这种操作符会影响结果。放在变量前面代表先执行加减操作然后再来运用变量的值。前置时只有下面这样的运算才有意义：

```
int x = 0;     int z = ++x;
```

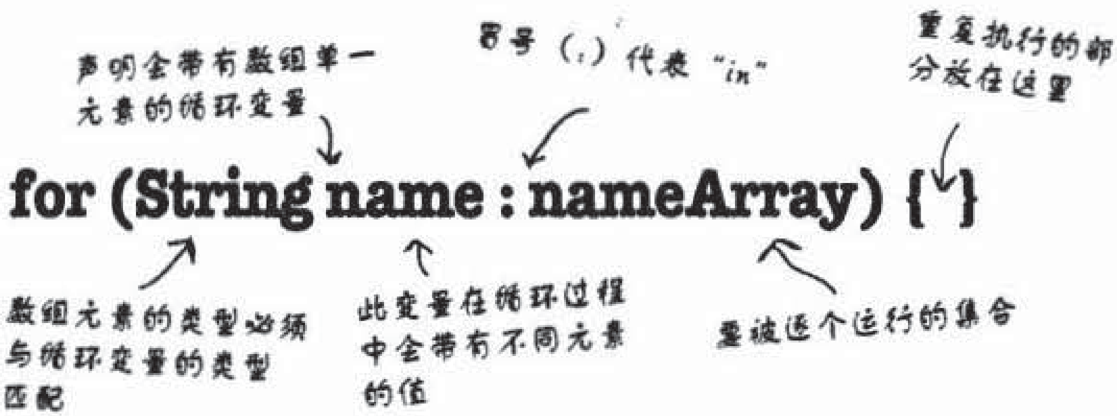
结果两个变量都是1。但是下面这一行会有不同的结果：

```
int x = 0;     int z = x++;
```

执行完x是1而z是0。z会被先指派x的值，然后才会执行递增x的操作。

加强版的for循环

从Java 5.0（Tiger）开始，Java语言就有称为加强版的for循环，它能够很容易地逐个运行数组或其他集合（collection）的元素。（下一章会讨论其他类型的集合）。这是个很好的强化功能，因为这是for循环很常见的用途。我们会在讨论非数组的集合时再次看到加强版的for循环。



上面这行程序以中文来说就是：“对nameArray中的每个元素执行一次”而编译器会这么认为：

（1）创建名称为name的String变量。

（2）将nameArray的第一个元素值赋给name。

（3）执行重复的内容。

（4）赋值给下一个元素name。

（5）重复执行至所有元素都被运行为止。

注意：在其他程序语言背景中，这种循环又称为"for each"或"for in"循环。

第一段：声明循环变量。

使用这个部分来声明与初始化用在循环内容的变量。循环过程中此变量所携带的值会有不同。此变量的类型必须要与数组元素匹配。

第二段：要运行的集合。

这必须是对数组或其他集合的引用。

将 String 转换成 int

```
int guess = Integer.parseInt(stringGuess);
```

玩家会在游戏提示时输入他的猜测。此猜测会以String的形式传递给checkYourself()方法。

但格子的位置是int，因此你无法将int与String进行比较。

举例来说，下面这程序无法运行：

String num - "2";

int x -2;

if (x --num) //这可不行！

编译此程序会得到错误信息：

```
operator -- cannot be applied
to int, java.lang.String
if (x == num) { }
       ^
```

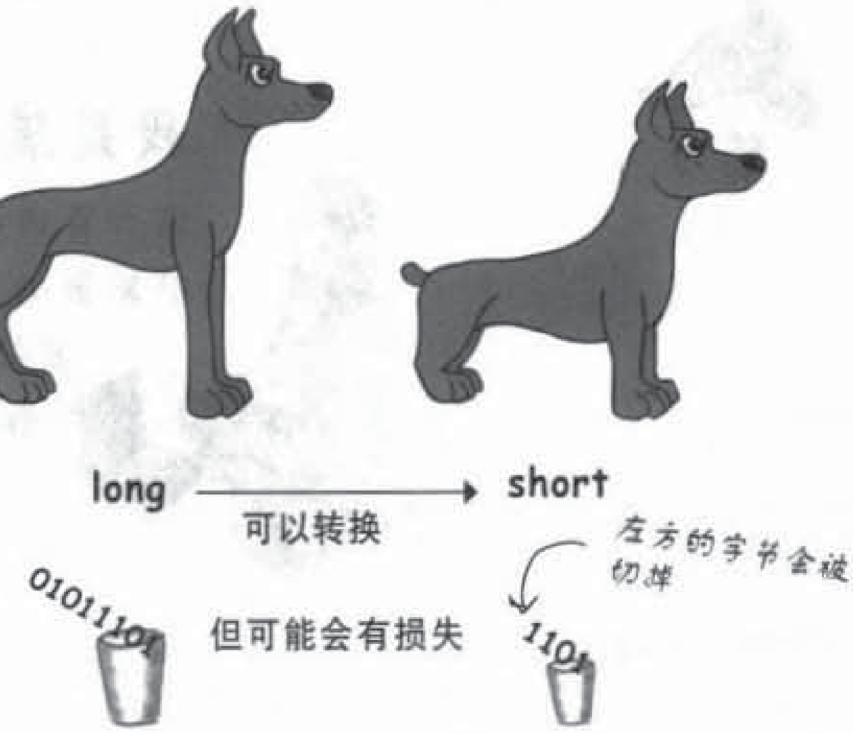
若要解决这类问题，我们必须要把String的"2"转换成int的2。Java内置有Integer这个类，它有一个方法能够将String所表示的数字转换成实际的数目。

Java内置的类 ↓　　取用String ↓

Integer.parseInt("3")

↗ 此方法知道如何将String解析成int

转换primitive主数据类型



我们在第3章讨论过各种primitive主数据类型的大小，以及小杯子无法装载大杯子的内容物的内容：

```
long y = 42;
int x = y;  // 不能通过编译
```

long比int大，且编译器无法确定long的内容是否可以截掉。若要强制编译器装，你可以使用cast运算符。

```
long y = 42;
int x = (int) y;
```

前置的类型转换会告诉编译器要将y的值裁剪成int的大小来赋值给x，但这个值可能会很诡异（见附录B）：

```
long y = 40002;
short x = (short) y;  // x的值会是-25534!
```

重点是这样可以通过编译程序。假如说你要取浮点数的整数值：

```
float f = 3.14f;
int x = (int) f;   // x的值会是3
```



我是编译器



这一页的Java程序代码代表一份完整的源文件。你的任务是要扮演编译器角色并判断程序输出会是哪一个？

```
class Output {

  public static void main(String [] args) {
    Output o = new Output();
    o.go();
  }

  void go() {
    int y = 7;
    for(int x = 1; x < 8; x++) {
      y++;
      if (x > 4) {
         System.out.print(++y + " ");
      }
      if (y > 14) {
        System.out.println(" x = " + x);
        break;
      }
    }
  }
}
```

File Edit Window Help OM

```
% java Output
12 14
```

或

File Edit Window Help Incense

```
% java Output
12 14 x = 6
```

或

File Edit Window Help Believe

```
% java Output
13 15 x = 6
```



排排看



右边是被打散的Java程序片段，你是否能够将它们重新排列以成为可以编译与执行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
x++;
```

```
if (x == 1) {
```

```
System.out.println(x + " " + y);
```

```
class MultiFor {
```

```
for(int  y = 4; y > 2; y--) {
```

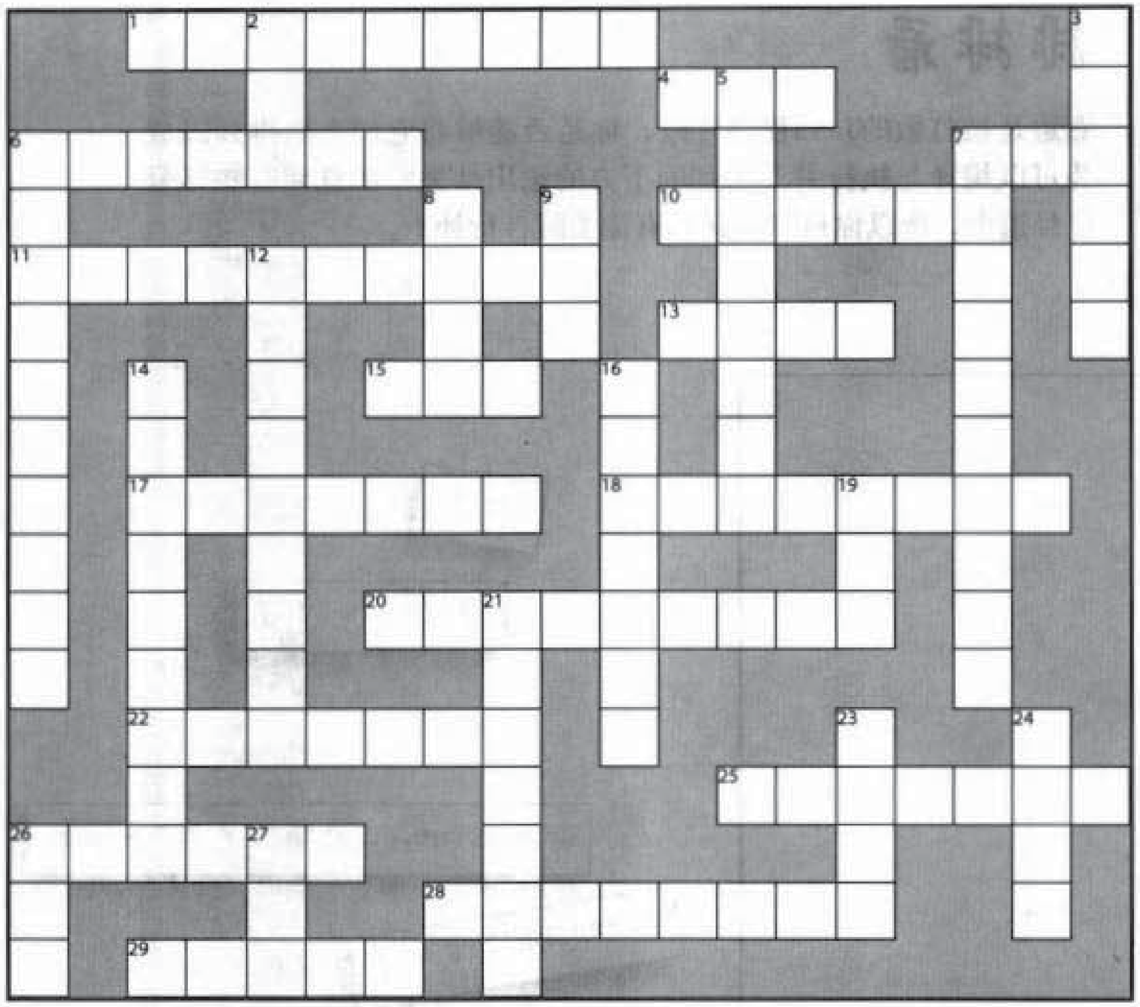
```
for(int x = 0; x < 4; x++) {
```

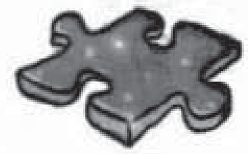
```
public static void main(String [] args) {
```

File Edit Window Help Raid

```
% java MultiFor
0 4
0 3
1 4
1 3
3 4
3 3
```

PDG





JavaCross

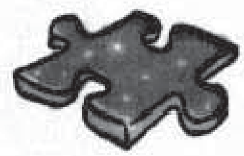
字谜游戏如何能够帮助学习Java呢？这些答案都与Java有关。

横排提示

1. Fancy computer word for build
4. Multi-part loop
6. Test first
7. 32 bits
10. Method's answer
11. Prepcode-esque
13. Change
15. The big toolkit
17. An array unit
18. Instance or local
20. Automatic toolkit
22. Looks like a primitive, but..
25. Un-castable
26. Math method
28. Converter method
29. Leave early

竖排提示

2. Increment type
3. Class's workhorse
5. Pre is a type of _____
6. For's iteration ______
7. Establish first value
8. While or For
9. Update an instance variable
12. Towards blastoff
14. A cycle
16. Talkative package
19. Method messenger (abbrev.)
21. As if
23. Add after
24. Pi house
26. Compile it and ____
27. ++ quantity



连连看

下面有一个Java小程序。其中有一段程序不见了。你的任务是找出下面所列出的程序段与相符的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

```
class MixFor5 {
  public static void main(String [] args) {
    int x = 0;
    int y = 30;
    for (int outer = 0; outer < 3; outer++) {
      for(int inner = 4; inner > 1; inner--) {
        [              ]

        y = y - 2;
        if (x == 6) {
          break;
        }
        x = x + 3;
      }
      y = y - 2;
    }
    System.out.println(x + " " + y);
  }
}
```

程序代码放这里

程序段代码：

- x = x + 3;
- x = x + 6;
- x = x + 2;
- x++;
- x--;
- x = x + 0;

相符的输出：

- 45 6
- 36 6
- 54 6
- 60 10
- 18 6
- 6 14
- 12 14

把可能的输出和程序代码连接起来。

PDG



解答

我是编译器

```
class Output {

  public static void main(String [] args) {
    Output o = new Output();
    o.go();
  }
  void go() {
    int y = 7;
    for(int x = 1; x < 8; x++) {
      y++;
      if (x > 4) {
         System.out.print(++y + " ");
      }
      if (y > 14) {
        System.out.println(" x = " + x);
        break;
      }
    }
  }
}
```

你记住break语句的要素了吗？
它会对输出产生什么样的影响？

```
File Edit Window Help MotorcycleMaintenance
% java Output
13 15 x = 6
```

排排看

```
class MultiFor {

  public static void main(String [] args) {

    for(int x = 0; x < 4; x++) {

      for(int y = 4; y > 2; y--) {
        System.out.println(x + " " + y);
      }

      if (x == 1) {
        x++;
      }
    }
  }
}
```

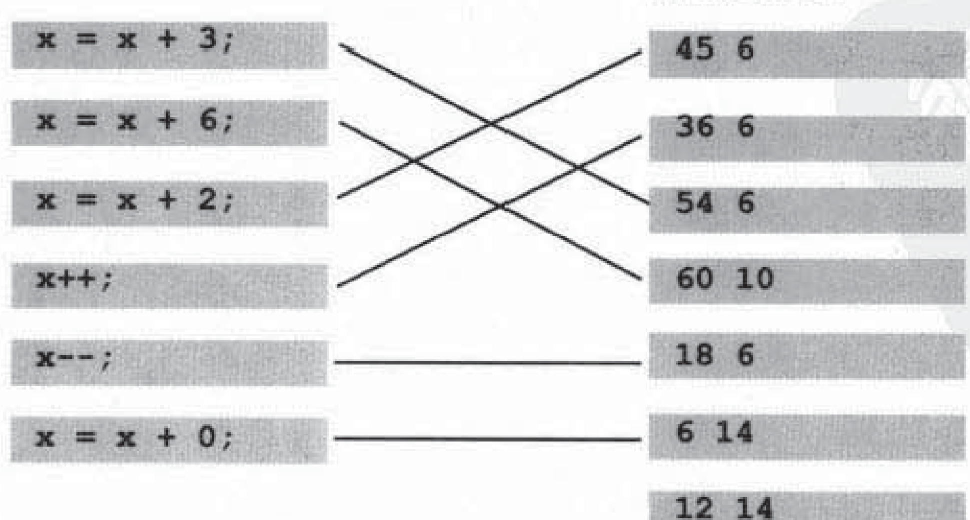
如果这个代码块出现在含y的for循环前，会发生什么？

```
File Edit Window Help Monopole
% java MultiFor
0 4
0 3
1 4
1 3
3 4
3 3
```

Java Cross



程序段代码：　　　　　　　　相符的输出：



6 认识Java的API

使用Java函数库



Java内置有数百个类。 如果你知道如何从通称Java API的Java的函数库中选找所需的功能，那么就不用重新发明轮子了。你可以把时间省下来做更有意义的事情。你听说过有些程序员上班总是迟到，而下班又很准时吗？因为他们使用Java API。读完这一章，你也可以这么混。核心Java函数库是由一堆等着被你当作组件使用的类集合而成的。你可以大量运用这些预先创建好的组件来写出你的程序。本书所附已经写好的程序代码让你不需要从无到有慢慢写出功能，但还是需要你自己输入进去。而Java API则根本不用输入，你只需要学会如何运用就好。

上一章的程序有个bug

正常情况

下面就是运行游戏，输入1，2，3，4，5，6的结果，看起来还不错！

完整的程序交互
（你的画面可能会不同）

```
File Edit Window Help Smile
%java SimpleDotComGame
enter a number  1
miss
enter a number  2
miss
enter a number  3
miss
enter a number  4
hit
enter a number  5
hit
enter a number  6
kill
You took 6 guesses
```

出现bug的情况

下面就是执行游戏，输入2，2，2的结果。

另一种结果
（哇……）

```
File Edit Window Help Faint
%java SimpleDotComGame
enter a number  2
hit
enter a number  2
hit
enter a number  2
kill
You took 3 guesses
```

在这个版本中，一旦你猜中一格，你就可以持续地猜同一格来击沉dot com！

发生了什么事？

```
public String checkYourself(String stringGuess) {

    int guess = Integer.parseInt(stringGuess);   ← 把String类型的变量转换成int类型

    String result = "miss";   ← 声明一个变量保存返回结果，默认情况下其值是"miss"

    for (int cell : locationCells) {

        if (guess == cell) {   ← 把用户猜测值与数组中的元素比较

            result = "hit";   ← 如果击中的话

            numOfHits++;

            break;   ← 跳出循环

        } // end if

    } // end for

    if (numOfHits == locationCells.length) {   ← 查看我们当前是否已经"dead"，并把结果改变成"kill"

        result = "kill";

    } // end if

    System.out.println(result);   ← 显示玩家的结果

    return result;   ← 把结果返回给调用的方法

} // end method
```

就是这里有问题。每当玩家猜中某一格时，我们就将计数器加数，而不管之前是否就已经被猜中。

我们需要一种机制来判别之前是否已经被猜中。

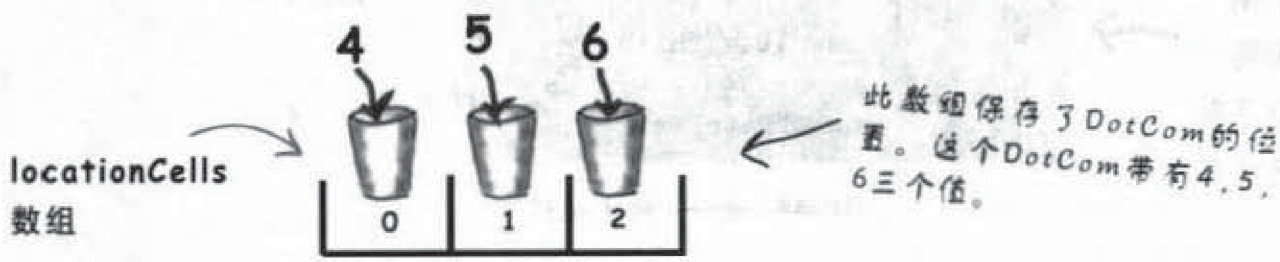
怎么解决？

我们需要一种方法来判别某个格子是否已经被猜中。让我们看看有哪些可能的解法。但首先得看我们已知的部分……

虚拟的行有7个格子，而DotCom会占有其中连续的3格。下面的虚拟列展示出占领 4, 5, 6 三格的 DotCom。

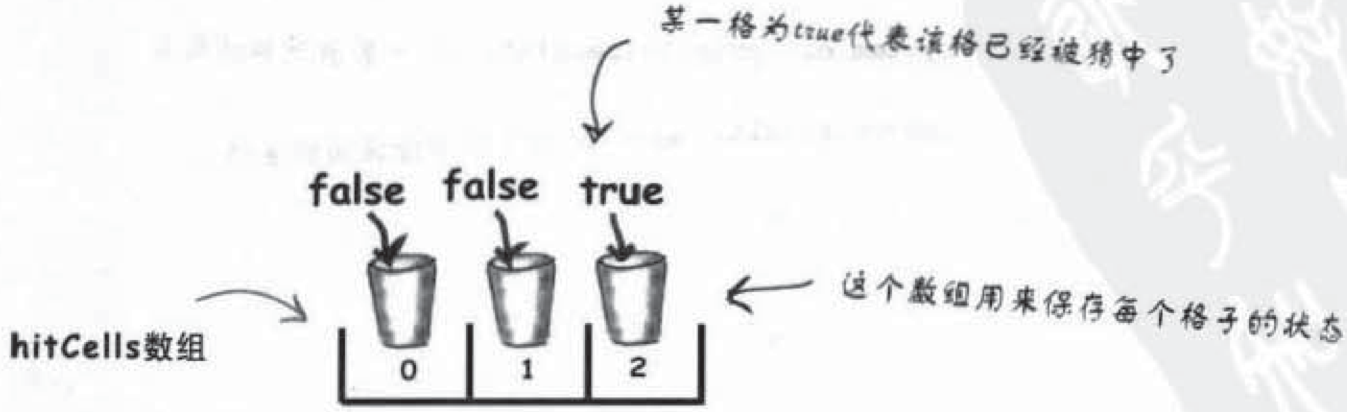


DotCom有个实例变量——一个int数组来保存DotCom对象的位置。



① 方案一

我们可以使用第二个数组。每当玩家猜中某一格时，我们就把相对的那一格设定成true，之后每次猜中都要检查是否在之前就已经被猜过了。

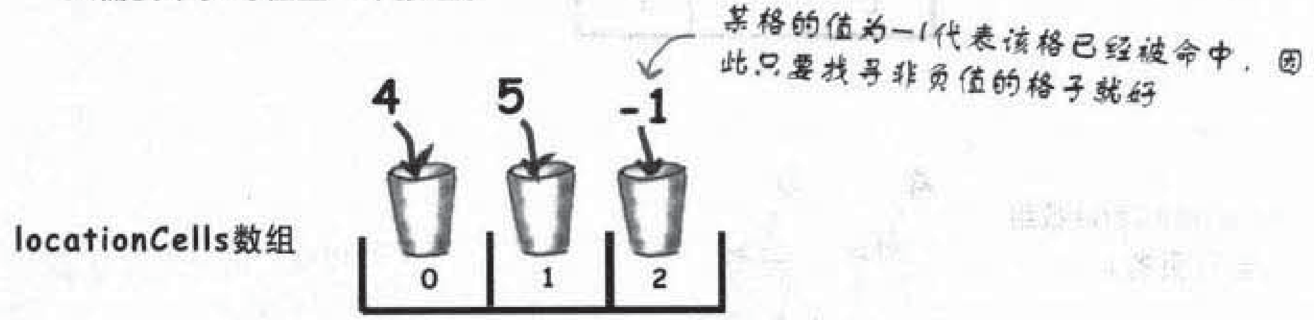


方案一不太优雅

方案一似乎太费时间了。那意味着每次玩家猜中某一格时，你都必须要改变第二个数组的状态，且在之前你还需要检查该数组来判别此格是否已经被命中。这是可行的方案没错，但一定还有更好的方法……

② **方案二**

我们可以只动用到原来的数组，办法是将任何被命中的格子改为-1。这样只需要维护与检查一个数组。



方案二比较好，但是还不够

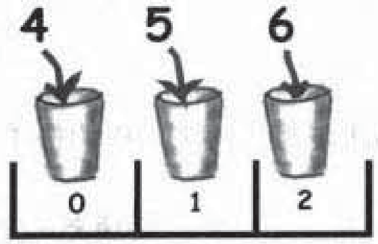
方案二比方案一好一点，但不是很有效率。就算有些格子已经被命中了，你还是得逐个搜寻数组中的那些格子。一定还有更好的办法……

伪码 测试码 真实码

③ 方案三

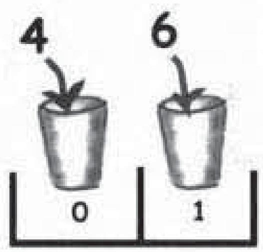
在命中某个格子时就把它删除掉，因此格子就会越来越少。但是数组无法改变大小，因此我们必须作出新的数组并拷贝旧数组的值。

locationCells数组（原来的）



数组的大小从3开始

locationCells数组（新拷贝的）



当5被命中的时候，我们就拷贝剩下的格子到新的数组中

如果数组能够缩小的话，方案三会更好，如此就不用拷贝并重新赋值引用。

原始的伪码：

```
REPEAT with each of the location cells in the int array
    // COMPARE the user guess to the location cell
    IF the user guess matches
        INCREMENT the number of hits
        // FIND OUT if it was the last location cell:
        IF number of hits is 3, RETURN "kill"
        ELSE it was not a kill, so RETURN "hit"
        END IF
    ELSE user guess did not match, so RETURN "miss"
    END IF
END REPEAT
```

如果能够改成这样会更好：

```
REPEAT with each of the remaining location cells
    // COMPARE the user guess to the location cell
    IF the user guess matches
        REMOVE this cell from the array
        // FIND OUT if it was the last location cell:
        IF the array is now empty, RETURN "kill"
        ELSE it was not a kill, so RETURN "hit"
        END IF
    ELSE user guess did not match, so RETURN "miss"
    END IF
END REPEAT
```


如果能够找到一种数组会在删除掉某些元素时自动缩小就好了。这样就不必检查所有的元素，只要查询它是否带有寻找中的值就好。若它还能够让你取出数据而不必管理集合的细节会更好！

别再幻想了

其实真的有这样的集合，但它不是数组，而是个ArrayList。它是Java函数库中的另一个类。

Java标准版本（除非你用的是小型装置的Micro Edition，不然你用的就是此版）带有数百个预先创建好的类。这就像我们写好给你的程序代码一样，只是说它们已经被编译过了。

这意味着你不用自行输入。

用就对了！

这只是Java函数库数百个类中的一个。

你可以把它们当作自己写的一样运用。

ArrayList

add(Object elem)
向list中加入对象参数

remove(int index)
在索引参数中移除对象

remove(Object elem)
移除该对象

contains(Object elem)
如果和对象参数匹配返回“true”

isEmpty()
如果list中没有元素返回“true”

indexOf(Object elem)
返回对象参数的索引或-1

size()
返回list中元素的个数

get(int index)
返回当前索引参数的对象

注意：实际上add(Object elem)这个方法比我们此处列出来的更为怪异，这个部分会在本书稍后的章节中解释。现在先当作就是这样。

这只是ArrayList的部分样本而已，实际更为复杂。

ArrayList 的操作

先别管这个 <括号>，这代表创建出Egg类型的List

① 创建

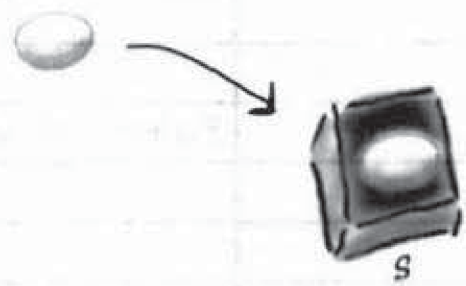
```
ArrayList<Egg> myList = new ArrayList<Egg>();
```

新的ArrayList对象会创建在堆上

② 加入元素

```
Egg s = new Egg();

myList.add(s);
```

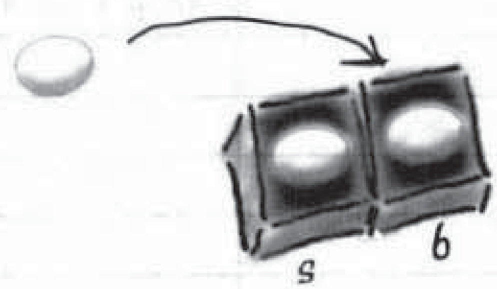


此ArrayList会产生出一个“盒子”来放Egg对象

③ 再加入元素

```
Egg b = new Egg();

myList.add(b);
```



此ArrayList会再弄出一个“盒子”来放新的Egg对象

④ 查询大小

```
int theSize = myList.size();
```

因为myList有两个元素，size()会返回2

⑤ 查询特定元素

```
boolean isIn = myList.contains(s);
```

因为myList带有s所引用的Egg对象，所以此方法会返回true

⑥ 查询特定元素的位置

```
int idx = myList.indexOf(b);
```

ArrayList为零基的，所以b引用的对象是第二个对象，而indexOf()会返回1。

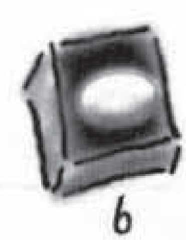
⑦ 判断集合是否为空

```
boolean empty = myList.isEmpty();
```

因为不是空的，isEmpty()会返回false

⑧ 删除元素

```
myList.remove(s);
```



注意，它被缩小了！

Sharpen your pencil

观察左方的 ArrayList 程序代码，然后将使用一般数组的相同功能程序代码填入右方空格中。这不是个测验，只是要让你自己动脑思考一下。

ArrayList数组	一般数组
ArrayList<String> myList = new ArrayList<String>();	String [] myList = new String[2];
String a = new String("whoohoo");	String a = new String("whoohoo");
myList.add(a);	
String b = new String("Frog");	String b = new String("Frog");
myList.add(b);	
int theSize = myList.size();	
Object o = myList.get(1);	
myList.remove(1);	
boolean isIn = myList.contains(b);	

there are no Dumb Questions

问：我怎么才能知道 ArrayList 的存在？

答： 这个问题其实是在问“我怎么知道API里面有什么？”。这对于能否成为优秀的Java程序员来说是非常关键的，同时也是让你在不用花太多时间的情况下依然能够创建出应用程序的重点所在。

简单地说就是多花点时间，而更详细的说明会在本章后面的内容中介绍。

问：这个回答等于没有回答。我不仅需要知道Java API带有ArrayList，更重要的是我要知道ArrayList可以解决我的问题！所以我要怎样才能知道有哪些 API 可以解决哪些问题？

答： 年轻人，我能感受到你的热情正在燃烧。如果你能够有耐心一点的话，再过几页我们就会开始讨论这个问题。



Java Exposed

本周的来宾：ArrayList

HeadFirst：ArrayList其实很像数组吧？

ArrayList：对不起，我是个对象！

HeadFirst：如果我没有记错的话，数组也是对象吧。它也是保存在堆上啊。

ArrayList：没错，但数组总是想要当个ArrayList。我们都知道对象有状态与行为对吧？但是你有对数组调用过方法吗？

HeadFirst：是啊，但是要调用什么功能？我只需要调用放进数组的东西就好，何必调用数组的功能呢？

ArrayList：是吗？你能从数组中删除元素吗？你在哪里学的Java啊？

HeadFirst：我可以用Dog d = dogArray[1]来取出元素，不是吗？

ArrayList：我不是说提取出元素，我说的是删除掉元素，听不懂吗？你那样做只是让d变量引用到数组中的元素所引用的相同对象而已。

HeadFirst：呃……我听错了。我是没有办法删除元素。

ArrayList：我可是个高级类呢，所以我可以确实地将引用删除掉，还能够动态地改变我的大小。

HeadFirst：我真不想继续讨论这个，但是外面有人放话说你只不过是个装模作样的无效率数组罢了。甚至有人还说你根本无法保存primitive主数据类型。那不是很不方便吗？

ArrayList：这种谎言你也相信？我不是没有效率的数组好不好，我承认有些情况下单纯的数组会比较快，但是谁会为了这一点点效能而放弃了一堆写好又有威力的功能？还有，看看我的弹性再说吧。要保存primitive主数据类型吗？没问题，用个包装类把primitive主数据类型包装起来不就得了吗？甚至在Java 5.0版本上，包装工作会自动进行。我会承认在运用primitive主数据类型的时候数组会比ArrayList快，拜托，现在还有谁在用primitive主数据类型？啊……几点了？我该去健身房了，下次再聊吧！

ArrayList与数组的比较

ArrayList	一般数组
ArrayList<String> myList = new ArrayList<String>();	String [] myList = new String[2];
String a = new String("whoohoo");	String a = new String("whoohoo");
myList.add(a);	myList[0] = a;
String b = new String("Frog");	String b = new String("Frog");
myList.add(b);	myList[1] = b;
int theSize = myList.size();	int theSize = myList.length;
Object o = myList.get(1);	String o = myList[1];
myList.remove(1);	myList[1] = null;
boolean isIn = myList.contains(b);	boolean isIn = false; for (String item : myList) { if (b.equals(item)) { isIn = true; break; } }

这里开始真的很不一样……

在使用ArrayList时，你只是在运用ArrayList类型的对象，因此就跟运用其他的对象一样，你会使用“.”运算符来调用它的方法。

使用数组时，你会以特殊的数组语法来操作。这样的语法只能用在数组上。虽然说数组也是对象，但是它有自己的规则，你无法调用它的方法，最多只能存取它的length实例变量。

PDG

比较 ArrayList 与一般数组

一般数组在创建时就必须确定大小

但对于ArrayList来说，你只需要创建出此类型的对象就行。它不需要指定大小，因为它会在加入或删除元素时自动地调整大小。

```
new String[2]
```

指定大小

```
new ArrayList<String>()
```

不需要指定大小

存放对象给一般数组时必须指定位置

（必须要指定介于0到比length小1之间的数字）

```
myList[1] = b;
```

指定索引值

如果索引值超越了数组的限制（例如说声明大小为2的数组，然后指派索引值3），程序会在执行期出现错误。

使用ArrayList时，你可以用add(Int, Object)这个形式的方法来指定索引值，或者使用add(Object) 的形式来给它自行管理大小。

```
myList.add(b);
```

不需指定索引值

一般数组使用特殊的语法

但ArrayList是个普通对象，所以不会有特殊的语法。

```
myList[1]
```

[方括号] 是只用在数组上的特殊语法

在Java 5.0中的ArrayList是参数化的（parameterized）

虽然我们说ArrayList不像一般数组有特殊的语法，但是它们在Java 5.0中有比较特殊的东西——参数化类型。

```
ArrayList<String>
```

<String>是类型参数。这代表String的集合，就像说ArrayList<Dog>代表Dog的集合

在Java 5.0之前是无法声明出要存放于ArrayList中元素的类型，它只会是异质对象的集合。现在我们就能用上面列出的语法来声明对象的类型。我们会在讨论Collection的章节对参数化类型作更进一步的探讨。

伪码 测试码 真实码

修改DotCom程序代码

这是有问题的版本：

我们把类的名称从SimpleDotCom改成DotCom，但是程序代码与上一章相同

```
public class DotCom {

   int[] locationCells;
   int numOfHits = 0;

   public void setLocationCells(int[] locs) {
      locationCells = locs;
   }

   public String checkYourself(String stringGuess) {
      int guess = Integer.parseInt(stringGuess);
      String result = "miss";

      for (int cell : locationCells) {
        if (guess == cell) {

          result = "hit";
          numOfHits++;

          break;
        }
      } // out of the loop

      if (numOfHits == locationCells.length) {
        result = "kill";
      }
      System.out.println(result);
        return result;
   } // close method
} // close class
```

有问题的部分，没有考虑到重复命中同一个格子的问题

伪码 测试码 真实码

全新配方的DotCom类

Now with ArrayList power!

```
import java.util.ArrayList;      // 先不管这一行，稍后说明

public class DotCom {

   private ArrayList<String> locationCells;
   // private int numOfHits;
   // 不需要上面这一行了！        // 将带有String的数组改成承载String的ArrayList

   public void setLocationCells(ArrayList<String> loc) {
      locationCells = loc;
   }                             // 参数有了新名称

   public String checkYourself(String userInput) {

      String result = "miss";    // 判别玩家猜测值是否有出现在ArrayList中，没有的话会返回-1

         int index = locationCells.indexOf(userInput);

         if (index >= 0) {       // 如果索引值大于或等于0，命中！

             locationCells.remove(index);   // 删除已经命中的格子

             if (locationCells.isEmpty()) {  // 如果全部命中清空，那就表示击沉了！
                result = "kill";
             } else {
                result = "hit";
             } // close if

         } // close outer if

      return result;
   } // close method
} // close class
```

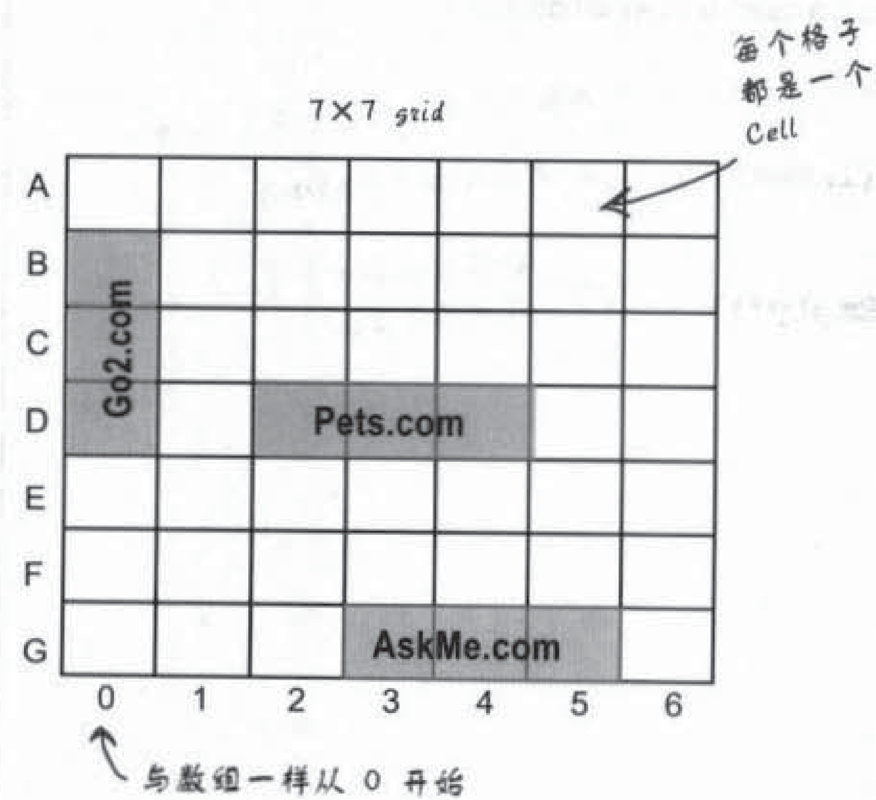
开发真正的“Sink a Dot Com”游戏

我们已经创建过简单版的，现在要进行豪华版的开发工作。相对于简单的横列，我们要使用方阵，并且现在有3个DotCom而不是只有一个而已。

游戏目标：以最少的猜测次数打掉计算机所安排的达康公司（Dot Com）网站。计算机会根据你的表现来评分。

初始设置：程序启动后，计算机会在虚拟的7×7方格上安排3个达康网站。安排完成后，游戏会要求你开始猜坐标。

进行游戏：因为我们还没有学到图形接口的程序设计，所以这一版会在命令行上进行。计算机会提示你输入所猜测的位置（格子），你会输入“A3”或“C5”等。计算机会回给你命中“Hit”、没中“Miss”或击沉“sunk”等回应。当你清光所有的达康时，游戏会列出你的分数并结束。



你会创建一个攻击网站游戏，它有7×7的格子与3间达康公司。每个达康网站占用3个格子。让我们重现网络大崩盘噩梦吧！

游戏进行中的画面

```
File Edit Window Help Sell
%java DotComBust
Enter a guess  A3
miss
Enter a guess  B2
miss
Enter a guess  C4
miss
Enter a guess  D2
hit
Enter a guess  D3
hit
Enter a guess  D4
Ouch! You sunk Pets.com   : (
kill
Enter a guess  B4
miss
Enter a guess  G3
hit
Enter a guess  G4
hit
Enter a guess  G5
Ouch! You sunk AskMe.com   : (
```

要改变什么？

有3个类需要修改：DotCom（以前叫做SimpleDotCom）、游戏的类（DotComBust）与辅助性类（现在先不管）。

A DotCom类

❍ **增加名称变量**

用来保存DotCom的名字，例如说“Pets.com”与“Go2.com”等。如此你就可以在它们被击沉时列出名字。

B DotComBust类(the game)

❍ **创建出3个DotCom**

❍ **指定DotCom的名称**

对每个DotCom的setter调用以设定name这个实例变量。

继续DotComBust类……

❍ **将DotCom放在方阵上**

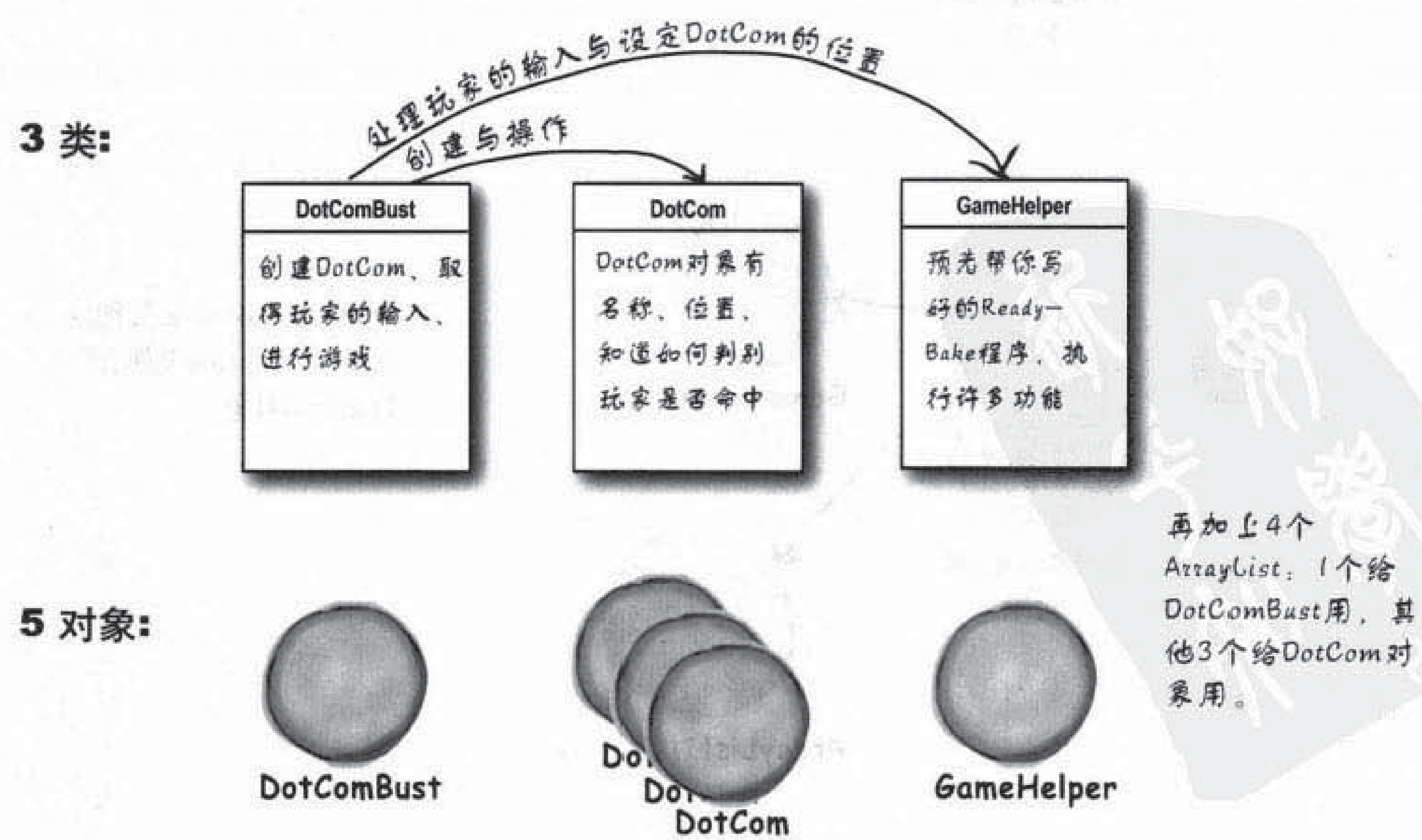
这个步骤会比简单版更为复杂，因为要考虑到重叠等问题。由于我们不想在这个类中引进复杂的数学问题，所以会把指定DotCom位置的算法放在GameHelper这个已经写好的辅助性类中。

❍ **每次猜测要检查3个DotCom**

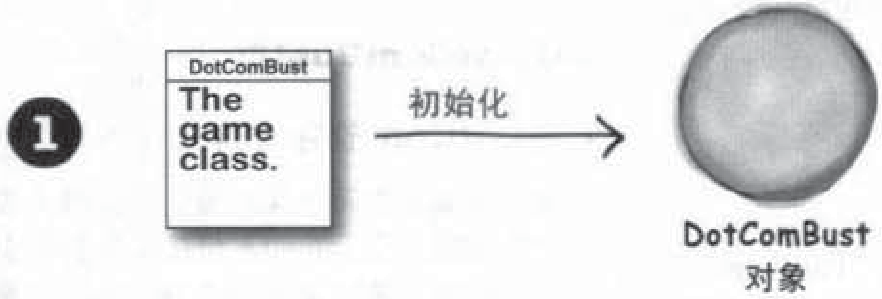
❍ **击沉3个DotCom后才能结束游戏**

❍ **脱离main()**

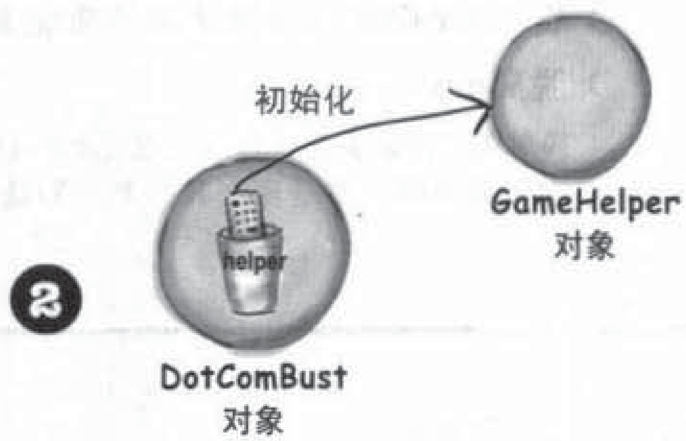
简单版之所以会把程序代码放在main()中是因为它是简单版，但是现在我们要写的是豪华复杂版。



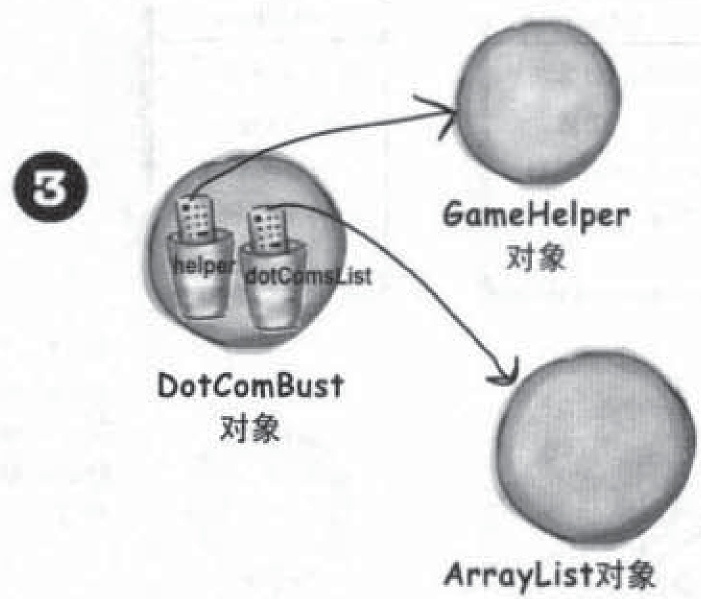
谁在何时做了什么？



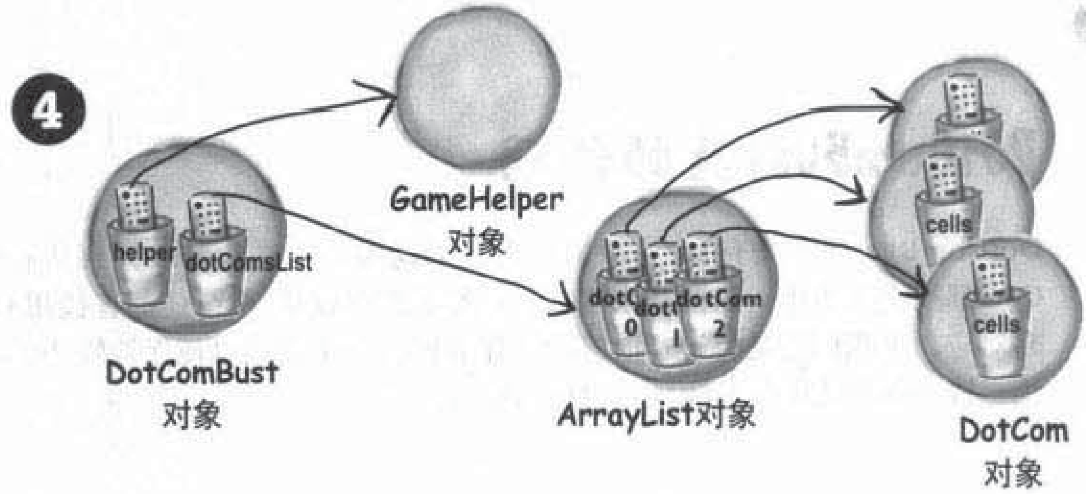
DotComBust这个类的main()方法会初始化DotComButs对象



DotComBust对象会初始化GameHelper的实例来帮助执行某些功能



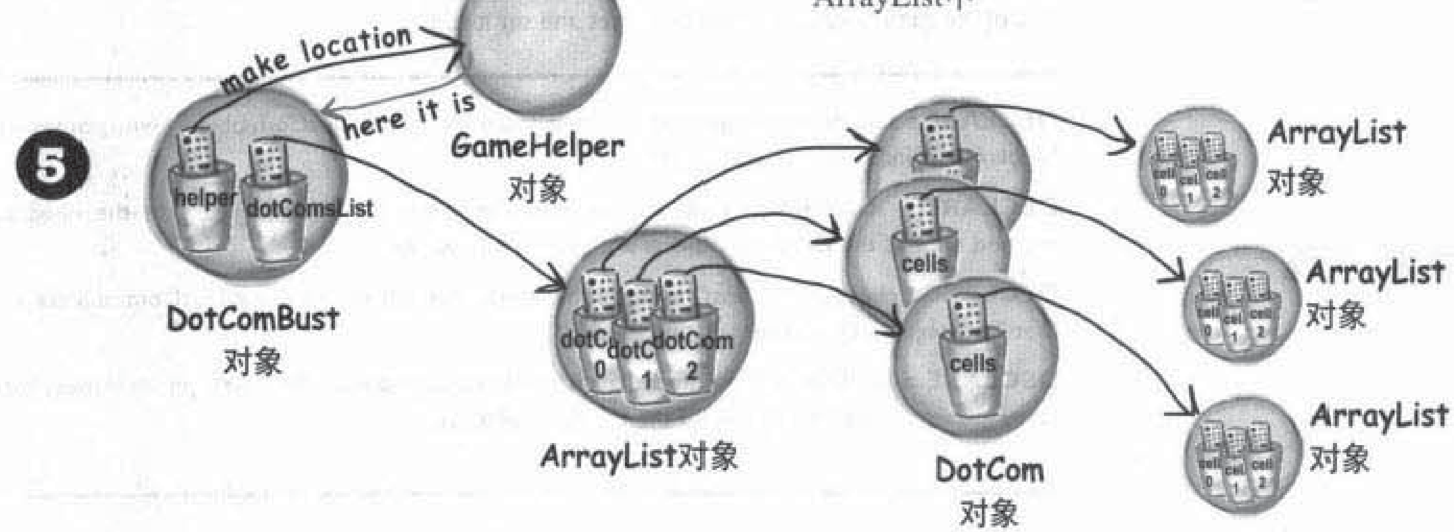
DotComBust对象会初始化1个ArrayList来保存3个DotCom对象



DotComBust对象创建出3个DotCom对象放到ArrayList中

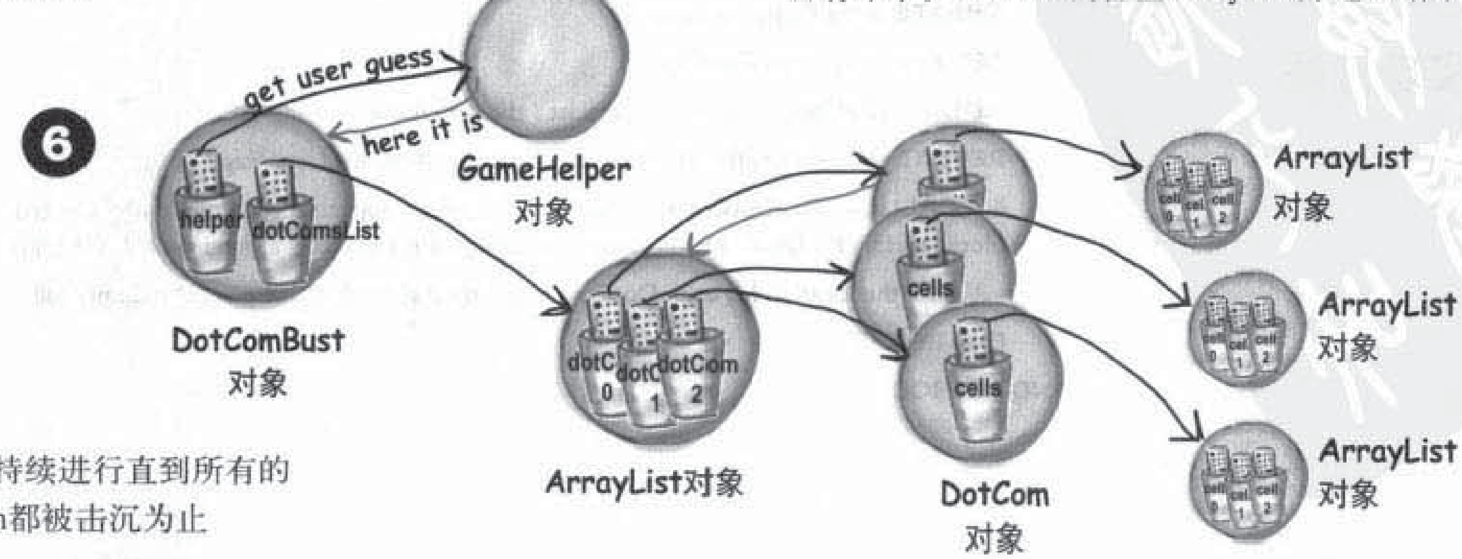
DotComBust对象询问helper对象来设定DotCom对象的位置

DotComBust对象赋值DotCom对象的位置，而DotCom对象则会把自己的3格放在独立的ArrayList中



DotComBust对象从helper对象取得玩家的猜测值

DotComBust对象逐个要求DotCom检查猜测值是否有命中。DotCom会检查ArrayList并返回结果



游戏会持续进行直到所有的DotCom都被击沉为止

伪码 测试码 真实码

DotComBust
GameHelper helper ArrayList dotComsList int numOfGuesses
setUpGame() startPlaying() checkUserGuess() finishGame()

DotComBust类的伪码

DotComBust这个类有3个主要的任务：启动游戏，进行游戏直到所有的Dot-Com都被击沉为止，以及结束游戏。虽然我们可以将这3个任务直接用3个方法来对应，但我们还是将进行游戏的工作分割成两个方法以便保持较小的功能模块。较小的方法比较好测试、除错与修改。

声明变量

DECLARE and instantiate the GameHelper instance variable, named helper.

DECLARE and instantiate an ArrayList to hold the list of DotComs (initially three) Call it dotComsList.

DECLARE an int variable to hold the number of user guesses (so that we can give the user a score at the end of the game). Name it numOfGuesses and set it to 0.

声明方法

DECLARE a setUpGame() method to create and initialize the DotCom objects with names and locations. Display brief instructions to the user.

DECLARE a startPlaying() method that asks the player for guesses and calls the checkUserGuess() method until all the DotCom objects are removed from play.

DECLARE a checkUserGuess() method that loops through all remaining DotCom objects and calls each DotCom object's checkYourself() method.

DECLARE a finishGame() method that prints a message about the user's performance, based on how many guesses it took to sink all of the DotCom objects.

实现方法

METHOD: ***void setUpGame()***

// make three DotCom objects and name them

CREATE three DotCom objects.

SET a name for each DotCom.

ADD the DotComs to the dotComsList (the ArrayList).

REPEAT with each of the DotCom objects in the dotComsList array

CALL the placeDotCom() method on the helper object, to get a randomly-selected location for this DotCom (three cells, vertically or horizontally aligned, on a 7 X 7 grid).

SET the location for each DotCom based on the result of the placeDotCom() call.

END REPEAT

END METHOD

伪码 测试码 **真实码**

实现方法（续）：

METHOD: ***void startPlaying()***

REPEAT while any DotComs exist

GET user input by calling the helper getUserInput() method

EVALUATE the user's guess by checkUserGuess() method

END REPEAT

END METHOD

METHOD: ***void checkUserGuess(String userGuess)***

// find out if there's a hit (and kill) on any DotCom

INCREMENT the number of user guesses in the numOfGuesses variable

SET the local result variable (a String) to "miss", assuming that the user's guess will be a miss.

REPEAT with each of the DotObjects in the dotComsList array

EVALUATE the user's guess by calling the DotCom object's checkYourself() method

SET the result variable to "hit" or "kill" if appropriate

IF the result is "kill", **REMOVE** the DotCom from the dotComsList

END REPEAT

DISPLAY the result value to the user

END METHOD

METHOD: *void finishGame()*

DISPLAY a generic "game over" message, then:

IF number of user guesses is small,

DISPLAY a congratulations message

ELSE

DISPLAY an insulting one

END IF

END METHOD

Sharpen your pencil

我们要如何将伪码转换成最终的程序代码？我们要先从测试代码开始，然后逐步地创建与测试方法。本书不会持续地列出测试码，所以你应该自行决定要如何测试这些方法。现在问题来了，你应该要先编写与测试哪个方法呢？尝试看看你是否能够作出测试用的程序代码。看别人写的程序代码或伪码是一回事，自己写出来又是另外一回事。

伪码 | 测试码 | **真实码**

```
import java.util.*;
public class DotComBust {

  ① private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

  private void setUpGame() {
     // first make some dot coms and give them locations
     DotCom one = new DotCom();              ②
     one.setName("Pets.com");
     DotCom two = new DotCom();
     two.setName("eToys.com");
     DotCom three = new DotCom();
     three.setName("Go2.com");
     dotComsList.add(one);
     dotComsList.add(two);
     dotComsList.add(three);

     System.out.println("Your goal is to sink three dot coms.");   ③
     System.out.println("Pets.com, eToys.com, Go2.com");
     System.out.println("Try to sink them all in the fewest number of guesses");

     for (DotCom dotComToSet : dotComsList) {   ④
        ArrayList<String> newLocation = helper.placeDotCom(3);   ⑤
        dotComToSet.setLocationCells(newLocation);   ⑥
     } // close for loop
  } // close setUpGame method

  private void startPlaying() {
     while(!dotComsList.isEmpty()) {   ⑦
       String userGuess = helper.getUserInput("Enter a guess");   ⑧
       checkUserGuess(userGuess);   ⑨
     } // close while
     finishGame();   ⑩
  } // close startPlaying method
```

Sharpen your pencil

自己批注乐趣多！

把下面的注释与对应的程序代码连接起来，在每个适当的注释前面写下该段程序代码的编号。

每个注释只会使用一次，所有的注释都会用到。

___ 要求DotCom的位置

___ 声明并初始化变量

___ 取得玩家输入

___ 对list中的每个DotCom重复一次

___ 列出简短的提示

___ 调用这个DotCom的setter方法来指派刚取得的位置

___ 调用checkUserGuess方法

___ 调用finishGame方法

___ 创建3个DotCom对象并指派名称并置入ArrayList

___ 判断DotCom的list是否为空

伪码 | 测试码 | **真实码**

```
private void checkUserGuess(String userGuess) {

    numOfGuesses++;  (11)
    String result  = "miss";  (12)

    for (DotCom dotComToTest : dotComsList) {  (13)
       result = dotComToTest.checkYourself(userGuess);  (14)
       if (result.equals("hit")) {
           break;  (15)
       }
       if (result.equals("kill")) {
           dotComsList.remove(dotComToTest);  (16)
           break;
       }
    } // close for
    System.out.println(result);  (17)
 } // close method

 private void finishGame() {
    System.out.println("All Dot Coms are dead! Your stock is now worthless.");
    if (numOfGuesses <= 18) {
       System.out.println("It only took you " + numOfGuesses + " guesses.");
       System.out.println(" You got out before your options sank.");
    } else {
       System.out.println("Took you long enough. "+ numOfGuesses + " guesses.");
       System.out.println("Fish are dancing with your options.");
    }
 } // close method

 public static void main (String[] args) {
    DotComBust game = new DotComBust();  (19)
    game.setUpGame();  (20)
    game.startPlaying();  (21)
 } // close method
}
```

(18) — finishGame() 中的 println 语句

千万不要翻到下一页！

只做完这些练习才准翻页。

答案就在下一页。 →

— 对list中所有的DotCom重复

— 列出结果

— 这家伙被挂掉了

— 列出玩家成绩

— 要求游戏对象启动

— 递增玩家猜测次数的计数

— 先假设没有命中

— 要求游戏对象启动游戏的主要循环

— 提前跳出循环

— 要求DotCom检查是否命中或击沉

— 创建游戏对象

伪码 | 测试码 | **真实码**

声明并初始化变量

```
import java.util.*;
public class DotComBust {

    private GameHelper helper = new GameHelper();
    private ArrayList<DotCom> dotComsList = new ArrayList<DotCom>();
    private int numOfGuesses = 0;

   private void setUpGame() {
      // first make some dot coms and give them locations
      DotCom one = new DotCom();
      one.setName("Pets.com");
      DotCom two = new DotCom();
      two.setName("eToys.com");
      DotCom three = new DotCom();
      three.setName("Go2.com");
      dotComsList.add(one);
      dotComsList.add(two);
      dotComsList.add(three);
```

创建3个DotCom对象并指派名称并置入ArrayList

```
      System.out.println("Your goal is to sink three dot coms.");
      System.out.println("Pets.com, eToys.com, Go2.com");
      System.out.println("Try to sink them all in the fewest number of guesses");
```

列出简短的提示

```
       for (DotCom dotComToSet : dotComsList) {
```

← 对list中所有的DotCom重复

```
          ArrayList<String> newLocation = helper.placeDotCom(3);
```

← 要求DotCom的位置

```
          dotComToSet.setLocationCells(newLocation);
```

← 调用这个DotCom的setter方法来指派刚取得的位置

```
      } // close for loop
   } // close setUpgame method

   private void startPlaying() {

     while(!dotComsList.isEmpty()) {
```

← 判断DotCom的list是否为空

```
        String userGuess = helper.getUserInput("Enter a guess");
```

← 取得玩家输入

```
        checkUserGuess(userGuess);
```

← 调用checkUserGuess方法

```
     } // close while
     finishGame();
```

← 调用finishGame方法

```
   } // close startPlaying method
```

伪码 测试码 **真实码**

```
private void checkUserGuess(String userGuess) {

    numOfGuesses++;                  // 递增玩家猜测次数的计数

    String result  = "miss";         // 先假设没有命中

    for (DotCom dotComToTest : dotComsList) {   // 对list中所有的DotCom重复

      result = dotComToTest.checkYourself(userGuess);  // 要求DotCom检查是否命中或击沉

      if (result.equals("hit")) {

           break;                    // 提前跳出循环
      }
      if (result.equals("kill")) {

           dotComsList.remove(dotComToTest);   // 这家伙被挂掉了
           break;
      }

    } // close for

    System.out.println(result);      // 列出结果
} // close method


private void finishGame() {          // 列出玩家成绩
    System.out.println("All Dot Coms are dead! Your stock is now worthless.");
    if (numOfGuesses <= 18) {
       System.out.println("It only took you " + numOfGuesses + " guesses.");
       System.out.println(" You got out before your options sank.");
    } else {
       System.out.println("Took you long enough. "+ numOfGuesses + " guesses.");
       System.out.println("Fish are dancing with your options");
    }
} // close method


public static void main (String[] args) {
    DotComBust game = new DotComBust();  // 创建游戏对象
    game.setUpGame();                    // 要求游戏对象启动
    game.startPlaying();                 // 要求游戏对象启动游戏的主要循环
} // close method
}
```

伪码 测试码 真实码

DotCom的最终版

```
import java.util.*;

public class DotCom {
   private ArrayList<String> locationCells;
   private String name;

   public void setLocationCells(ArrayList<String> loc) {
      locationCells = loc;
   }

      public void setName(String n) {
      name = n;
  }

   public String checkYourself(String userInput) {
      String result = "miss";
      int index = locationCells.indexOf(userInput);
      if (index >= 0) {
         locationCells.remove(index);

        if (locationCells.isEmpty()) {
            result = "kill";
            System.out.println("Ouch! You sunk " + name + "   : ( ");
        } else {
            result = "hit";
        }  // close if
      } // close if
       return result;

   } // close method
} // close class
```

DotCom的实例变量：
——保存位置的ArrayList
——DotCom 的名称

更新DotCom位置的setter方法

基本的setter方法

使用到indexOf()方法！如果玩家猜中的话，这个方法会返回它的位置。如果没有的话会返回-1

删除被猜中的元素

用这个方法来判别是否击沉

列出击沉的信息

返回状态

超强布尔表达式

到目前为止，我们在 if 测试或循环中所使用到的布尔表达式都很简单。接下来你会在预先写好的程序代码中看到一些更复杂的布尔表达式。现在先来看一下这些表达式。

“与”和“或”运算符(&&, ||)

假如说你正在编写chooseCamera()方法，它有一些选择相机的规则。或许你会想要选择介于$50与$1000之间的相机，但有时你会想要更精确的指定范围。例如说：

“如果价格范围在$300和$400之间，就选择X牌相机”

```
if (price >= 300 && price < 400) {
    camera = "X";
}
```

又假如说有10种不同品牌的相机可供选择，你有这样的逻辑来限制品牌：

```
if (brand.equals("A") || brand.equals("B") ) {
   // 执行A或B才能进行的工作
}
```

这样的布尔运算式会很大并且很复杂：

```
if ((zoomType.equals("optical") &&
    (zoomDegree >= 3 && zoomDegree <= 8)) ||
    (zoomType.equals("digital") &&
    (zoomDegree >= 5 && zoomDegree <= 12))) {
   // 执行适当的工作
}
```

技术上，你可能会搞不太清楚这些运算符的优先级。与其花时间研究这些规则，不如用括号来让程序代码更容易阅读。

“不等于”运算符 (!=和!)

假如说你有这样的规则：有一项规则仅适用于10台相机其中的1台。

```
if (model != 2000) {
   // 非 model 2000 的工作
}
```

或者是这样：

```
if (!brand.equals("X")) {
   // 非X牌的工作
}
```

短运算符 (&&, ||)

像&&与||，这些我们已经看过的运算符都称为短运算符。在&&表达式中，左右两边都为true这个表达式才会为true。因此，如果Java虚拟机发现左方的表达式为false，则它不需也不会去计算右方的算式才知道要返回false。||也有相同的特点。所以我们可以用下面这种方式来避免调用内容为null的指针变量的方法：

```
if (refVar != null &&
      refVar.isValidType() ) {
   // 执行有效变量的工作
}
```

长运算符 (&, |)

& 与 | 运算符使用在boolean表达式时会强制Java虚拟机一定要计算运算符两边的算式。但这两个运算符通常是用来作位的运算。



现成码

这是游戏会使用到的辅助性类。除了取得玩家输入的方法之外，这个类的另外一个最大作用是设置**DotCom**的位置。如果我是你的话，我会先不管这个类，只需要把它编写出来让程序能够编译就好。这个程序用到一些技巧使得它不是很容易理解。

注意：你可以把System.out.print这些行的注释去掉，这样会泄漏出位置帮你作弊或者方便测试。

```
import java.io.*;
import java.util.*;

public class GameHelper {

  private static final String alphabet = "abcdefg";
  private int gridLength = 7;
  private int gridSize = 49;
  private int [] grid = new int[gridSize];
  private int comCount = 0;

  public String getUserInput(String prompt) {
     String inputLine = null;
     System.out.print(prompt + "  ");
     try {
       BufferedReader is = new BufferedReader(
        new InputStreamReader(System.in));
       inputLine = is.readLine();
       if (inputLine.length() == 0 )  return null;
     } catch (IOException e) {
       System.out.println("IOException: " + e);
     }
     return inputLine.toLowerCase();
  }

 public ArrayList<String> placeDotCom(int comSize) {
   ArrayList<String> alphaCells = new ArrayList<String>();
    String [] alphacoords = new String [comSize];          // 保存字符串
    String temp = null;                                    // 临时字符串
    int [] coords = new int[comSize];                      // 现有字符串
    int attempts = 0;                                      // 目前测试的字符串
    boolean success = false;                               // 找到适合位置吗?
    int location = 0;                                      // 目前起点

    comCount++;                                            // 现在处理到第n个
    int incr = 1;                                          // 水平增量
    if ((comCount % 2) == 1) {                             // 如果是单数号的
      incr = gridLength;                                   // 垂直增量
    }

    while ( !success & attempts++ < 200 ) {                // 主要搜索循环
       location = (int) (Math.random() * gridSize);        // 随机起点
        //System.out.print(" try " + location);
       int x = 0;                                          // 第n个位置
         success = true;                                   // 假定成功
         while (success && x < comSize) {                  // 查找未使用的点
           if (grid[location] == 0) {                      // 如果没有使用
```



现成码

GameHelper类的程序代码（续）：

```
          coords[x++] = location;                       // 储存位置
          location += incr;                             // 尝试下一个点
          if (location >= gridSize){                    // 超出下边缘
            success = false;                            // 失败
          }
          if (x>0 && (location % gridLength == 0)) {    // 超出右边缘
            success = false;                            // 失败
          }
      } else {                                          // 找到已经使用的位置
          // System.out.print(" used " + location);
          success = false;                              // 失败
      }
    }
  }                                                     // while结束

  int x = 0;                                            // 将位置转换成字符串形式
  int row = 0;
  int column = 0;
  // System.out.println("\n");
  while (x < comSize) {
    grid[coords[x]] = 1;                                // 标识格子已用
    row = (int) (coords[x] / gridLength);               // 得到行的值
    column = coords[x] % gridLength;                    // 得到列的值
    temp = String.valueOf(alphabet.charAt(column));     // 转换成字符串

    alphaCells.add(temp.concat(Integer.toString(row)));
    x++;
    // System.out.print("  coord "+x+" = " + alphaCells.get(x-1));
  }

  // System.out.println("\n");

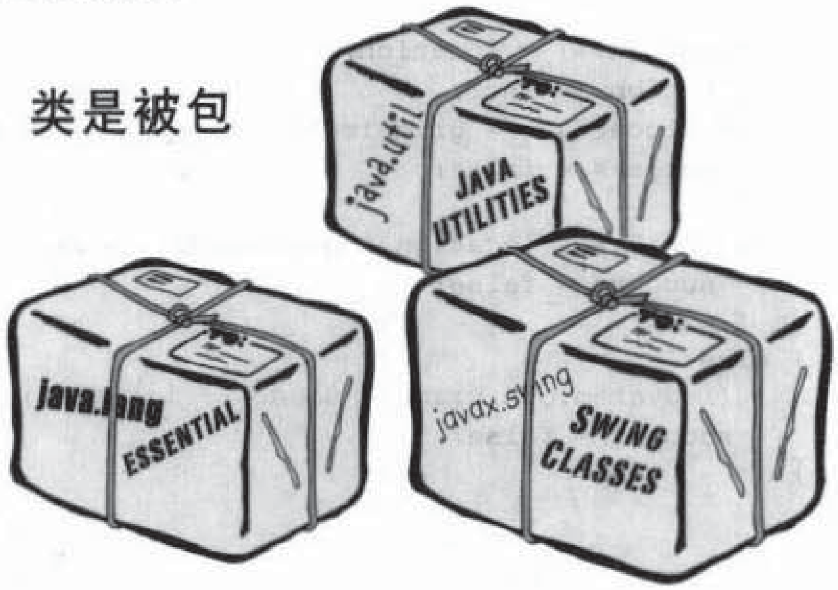
  return alphaCells;
 }
}
```

这一行会告诉你DotCom的确实位置

使用函数库（Java API）

我们已经通过了DotComBust游戏的开发过程。这得要感谢ArrayList的帮忙。现在该是来看如何运用Java函数的时候了。

在Java的API中，类是被包装在包中。



要使用API中的类，你必须知道它被放在哪个包中。

在Java函数库中的每个类都属于某个包。这些包都有个名字，像是javax.swing（里面带有很快就会遇到的Swing 接口类）。ArrayList 是放在 java.util 这个包中。顾名思义，java.util放了很多工具类。第16章会讨论有关包的细节，包括如何自制包。

使用来自API的类是很简单的。只要把它们当作是自己写的就好，但是其中还有一个很大的不同：在程序的某个地方你必须要指明函数库类的完整名称，也就是包的名称加上类的名称。

也许你还不清楚，但是实际上已经到了好几个来自 API 的类。像是System（System.out.println）、String与Math（Math.Random( )）都是属于java.lang这个包。

你必须指明程序代码中所使用到的类的完整名称*

ArrayList并不是它的全名，如同“Mike”也不是全名一样（除非是Madonna 或Cher这种）。完整的名称是这样的：

```
java.util.ArrayList
```

包的名称　　　　类的名称

你必须告诉Java想要使用的是哪一个ArrayList。有两种方法可以指定：

A IMPORT

放一个import述句在程序源文件的最前面：

```
import java.util.ArrayList;
public class MyClass {... }
```

或

B TYPE

或者在程序代码中打出全名。不管在哪里，只要有使用到就打出全名。

声明的时候：

```
java.util.ArrayList<Dog> list = new java.util.ArrayList<Dog>();
```

用在参数的时候：

```
public void go(java.util.ArrayList<Dog> list) { }
```

作为返回类型的时候：

```
public java.util.ArrayList<Dog> foo() {...}
```

*除非是来自于 java.lang 这个包中。

there are no Dumb Questions

问：为何需要全名？这就是包的由来吗？

答：包之所以很重要有3个原因。首先，它们可以帮助组织项目或函数库相对于一大堆零散的类，以功能来组织会比较好。

其次，包可以制造出名称空间，以便错开相同名称的类。例如说有好几个程序员都设计出Set这个类，我们就可以通过不同的包名称来分辨。

最后，包可以通过限制同一包之间的类才能相互存取以维护安全性。

问：那要如何防止包的名称也产生冲突？

答：如果每个程序员都遵循的话，Java的命名传统就能够防止发生这种事情。第16章会有更进一步的探讨。

这个“x”是从哪来的？（javax开头的包代表什么？）

在Java的早期两个版本中（1.02与1.1），所有随附于Java的类（也就是standard library）都是放在java开头的包中。例如java.lang、java.io、java.util等。

后来出现了一些没有包含在标准函数库中的包。这些被称为扩展的类有两种类型：标准的与非标准的。Sun所认可的称为standard extension，其余实验性质、预览版本或beta版的非标准类则不一定会被认可采用。

标准版的扩展都以javax作为包名称的开头。最早出现的是Swing函数库，它包含的数个包都是以javax.swing开头。从Java 1.2（又称为Java 2）开始，Swing就一并被包含在Java中。

每个人都认为这样很酷，因为如此一来就不必担心用户要如何安装与Swing有关之扩展的步骤。但这些包后来被认可成为标准的一部分，所以Sun在发行1.2版前将开头名称从javax换成了java。很多使用到Swing程序代码的书籍也就这样以新名称印刷发售。

但是非常多的开发者发现这会造成重大的社会写实悲剧，他们之前所写的每一个Swing程序都要跟着改写！想到有多少个import述句是以javax开头就让人痛不欲生……

在最后的关头，开发者终于说服Sun采用“管它的命名传统，先保护程序再说”的方法。所以现在你看到函数库中以javax开头的包就会知道它以前曾经是扩展，后来才取得一个标准名份的。

要点

- ArrayList是个Java API的类。
- 使用add( )来新增ArrayList的元素。
- 使用remove()来删除ArrayList中的元素。
- 要寻找某项元素的位置，使用indexOf()。
- 使用isEmpty()来判别ArrayList是否为空。
- 要取得ArrayList的大小，可以使用size()方法。
- 传统的数组可以用length这个变量取得大小。
- ArrayList会自动地调整大小。
- 你可以用参数类型来声明数组内容的类型，例如ArrayList<Button>会声明带有Button类型元素的ArrayList。
- 虽然ArrayList只能携带对象而不是primitive主数据类型，但编译器能够自动地将primitive主数据类型包装成Object以存放在ArrayList 中。
- 类会用包来组织。
- 类有完整的名称，那是由包的名称与类的名称所组成的。ArrayList事实上叫做java.util.ArrayList。
- 除了java.lang之外，使用到其他包的类都需要指定全名。
- 也可以在原始程序代码的最开始部分下import指令来说明所使用到的包。

there are no Dumb Questions

问： 使用import会把程序变大吗？编译过程会把包或类包进去吗？

答： 你一定是个C语言程序员。import与C的include并不相同。运用import只是帮你省下每个类前面的包名称而已。程序不会因为用了import而变大或变慢。

问： 那为何我不必import进String类或System类？

答： 要记得java.lang是个预先被引用的包。因为java.lang是个经常会用到的基础包，所以你可以不必指定名称。java.lang.String与java.lang.System是独一无二的class，Java会知道要去哪里找。

问： 我有需要将自己写的类包进包中吗？要怎么做？

答： 在实际应用中，你应该会把类包进包中。第16章会讨论这个问题。现在我们还不需要担心这个。



Make it Stick

玫瑰是红的，

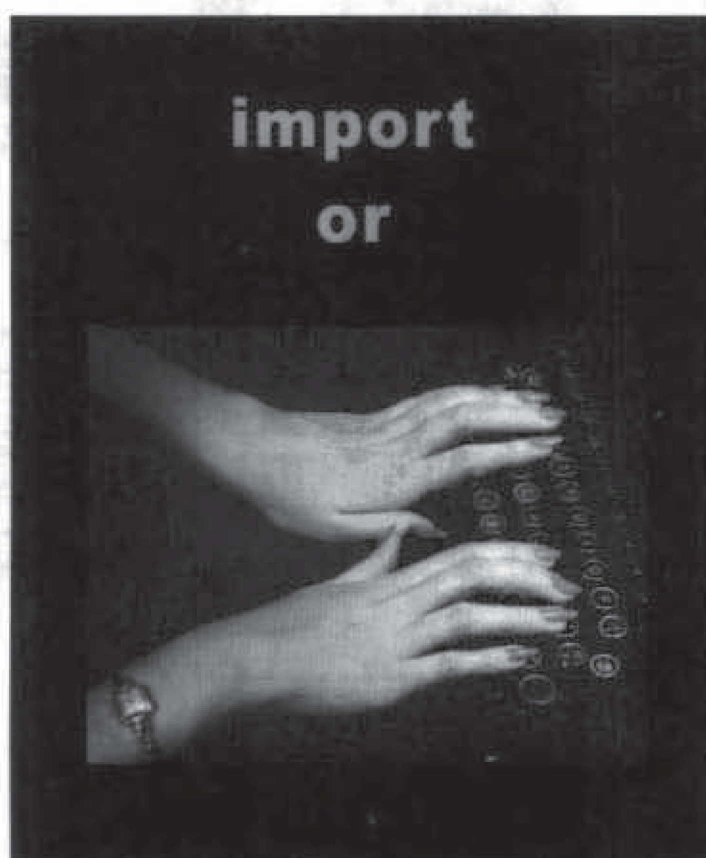
数学老师是常常请假的，

如果你不想多打字，

那就要import。

除了在java.lang这个包里面的类，要用到其他的类都要指定完整名称。如果怕麻烦的话，那就用import来帮助你。

怕你还没有记得，我们再说一次：



"很高兴知道在java.util这个包中有ArrayList。但是我自己要怎么找呢？"

——Julia，31岁，模特



如何查询API？

有两件事情你必须知道：

1. 库中有哪些类？
2. 找到类之后，你怎么知道它是做什么的？

1 查阅参考书



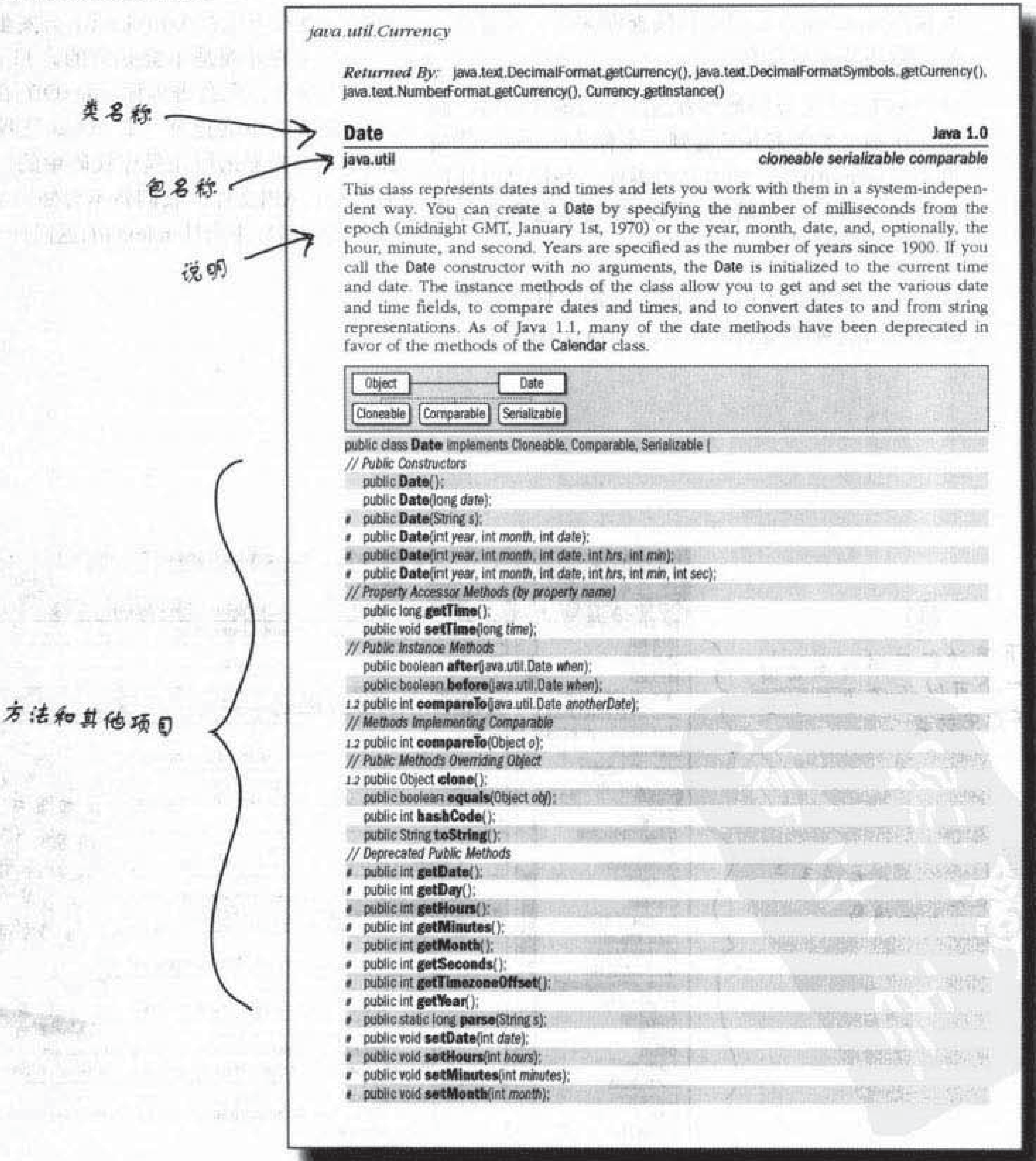
2 查阅HTML API文档



❶ 查阅参考书



随意翻阅参考书是找出Java 函数库有哪些可用类的最佳方式。遇到有意思的就多看几眼。



java.util.Currency

Returned By: java.text.DecimalFormat.getCurrency(), java.text.DecimalFormatSymbols.getCurrency(), java.text.NumberFormat.getCurrency(), Currency.getInstance()

Date — **Java 1.0**

java.util — ***cloneable serializable comparable***

This class represents dates and times and lets you work with them in a system-independent way. You can create a **Date** by specifying the number of milliseconds from the epoch (midnight GMT, January 1st, 1970) or the year, month, date, and, optionally, the hour, minute, and second. Years are specified as the number of years since 1900. If you call the **Date** constructor with no arguments, the **Date** is initialized to the current time and date. The instance methods of the class allow you to get and set the various date and time fields, to compare dates and times, and to convert dates to and from string representations. As of Java 1.1, many of the date methods have been deprecated in favor of the methods of the **Calendar** class.

Object — Date
Cloneable | Comparable | Serializable

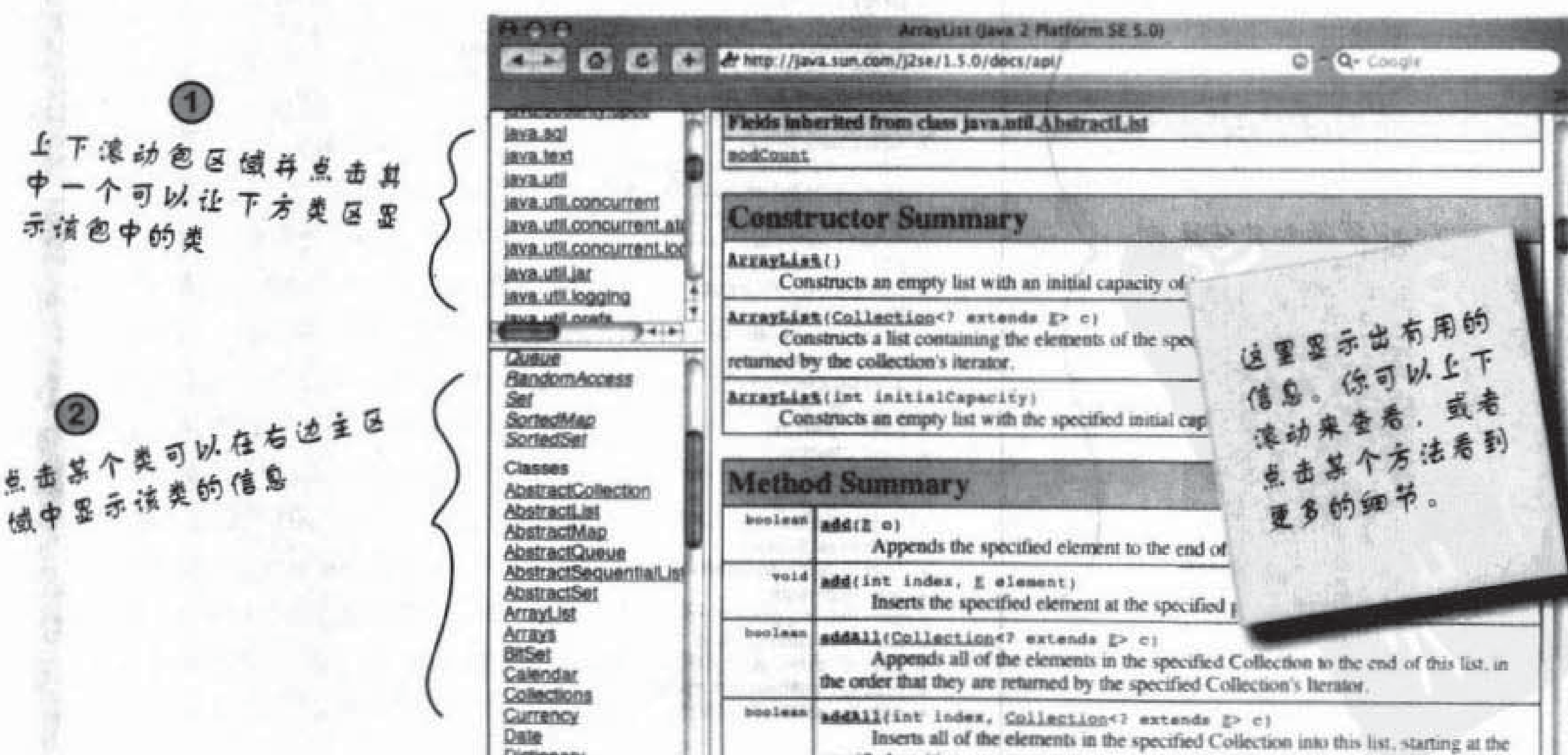
```
public class Date implements Cloneable, Comparable, Serializable {
// Public Constructors
    public Date();
    public Date(long date);
#   public Date(String s);
#   public Date(int year, int month, int date);
#   public Date(int year, int month, int date, int hrs, int min);
#   public Date(int year, int month, int date, int hrs, int min, int sec);
// Property Accessor Methods (by property name)
    public long getTime();
    public void setTime(long time);
// Public Instance Methods
    public boolean after(java.util.Date when);
    public boolean before(java.util.Date when);
1.2 public int compareTo(java.util.Date anotherDate);
// Methods Implementing Comparable
1.2 public int compareTo(Object o);
// Public Methods Overriding Object
1.2 public Object clone();
    public boolean equals(Object obj);
    public int hashCode();
    public String toString();
// Deprecated Public Methods
#   public int getDate();
#   public int getDay();
#   public int getHours();
#   public int getMinutes();
#   public int getMonth();
#   public int getSeconds();
#   public int getTimezoneOffset();
#   public int getYear();
#   public static long parse(String s);
#   public void setDate(int date);
#   public void setHours(int hours);
#   public void setMinutes(int minutes);
#   public void setMonth(int month);
```

❷ 查阅HTML API文档

Java有个很了不起的在线文件，它很奇怪地被称为Java API。它是Java 5标准版文档的一部分（之前叫做Java 2标准版5.0），并且与Java是分开下载的。如果你联上Internet，并且很有耐心，也可以直接在java.sun.com网站在线查询阅读。相信我，最后你还是会下载的。

这份API文件是查询类与方法细节的最佳参考。假如你在翻阅参考书并且看到一个称为Calendar的类出现在java.util中。书本会告诉你一些信息以让你知道这是否是你要用的，但是你还是会需要知道更多的相关细节。

例如，参考书会告诉你它的方法需要什么参数，以及返回何种类型的数据。以ArrayList为例，在参考书中你会看到有个称为indexOf()的方法，我们在DotCom这个类中用到它。但你所能知道的也只限于indexOf()取用一个对象参数并返回int类型的索引值，你还是必须要查出最重要的信息：如果对象没有出现在ArrayList中会发生什么事情？光看方法的外观是不会知道的。只有API文件会告诉你细节。它会告诉你indexOf()在找不到相符对象的情况下会返回－1。这就是我们如何知道ArrayList要怎么用在程序代码中的方式。如果没有参考过API文件，我们将不会知道ArrayList对于找不到的情况下会让indexOf()返回什么。





排排看

右边是被打散的Java程序片段，你是否能够将它们重新排列成为可以编译与运行并产生如同下方的输出结果？注意，你必须查询API文件以找出取用两个参数的add方法的说明。

add(int index, Object o)

```
a.remove(2);
```

```
printAL(a);
```

```
printAL(a);
```

```
a.add(0,"zero");
a.add(1,"one");
```

```
public static void printAL(ArrayList<String> al) {
```

```
if (a.contains("two")) {
   a.add("2.2");
}
```

```
a.add(2,"two");
```

```
public static void main (String[] args) {
```

```
System.out.print(element + "  ");
}
System.out.println(" ");
```

```
if (a.contains("three")) {
   a.add("four");
}
```

```
public class ArrayListMagnet {
```

```
if (a.indexOf("four") != 4) {
   a.add(4, "4.2");
}
```

```
}
```

```
}
```

```
}
```

```
import java.util.*;
```

```
printAL(a);
```

```
ArrayList<String> a = new ArrayList<String>();
```

```
for (String element : al) {
```

```
a.add(3,"three");
printAL(a);
```

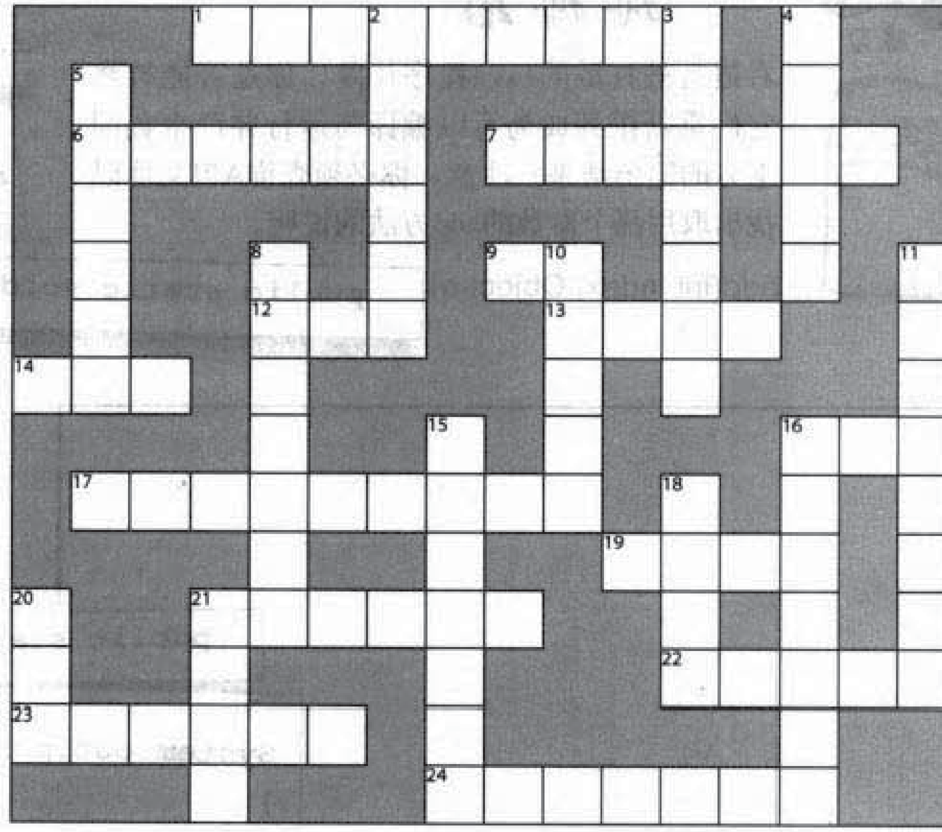
File Edit Window Help Dance

```
% java ArrayListMagnet
zero  one  two  three
zero  one  three  four
zero  one  three  four  4.2
zero  one  three  four  4.2
```



JavaCross 7.0

字谜游戏如何能够帮助学习Java呢？这些答案大部分都与Java以及ArrayList有关。



横排提示：

1. I can't behave
6. Or, in the courtroom
7. Where it's at baby
9. A fork's origin
12. Grow an ArrayList
13. Wholly massive
14. Value copy
16. Not an object
17. An array on steroids
19. Extent
21. 19's counterpart
22. Spanish geek snacks (Note: This has nothing to do with Java.)
23. For lazy fingers
24. Where packages roam

竖排提示：

2. Where the Java action is.
3. Addressable unit
4. 2nd smallest
5. Fractional default
8. Library's grandest
10. Must be low density
11. He's in there somewhere
15. As if
16. dearth method
18. What shopping and arrays have in common
20. Library acronym
21. What goes around

更进一步的提示：

Across
1. 8 varieties
7. Think ArrayList
16. Common primitive
21. Array's extent
22. Not about Java - Spanish appetizers

Down
2. What's overridable?
3. Think ArrayList
4. & 10. Primitive
16. Think ArrayList
18. He's making a ______

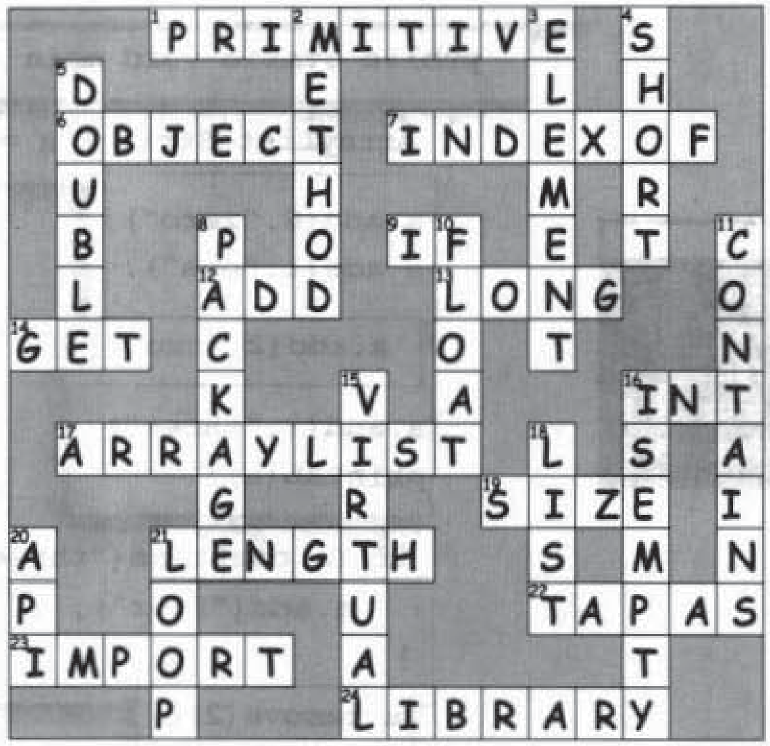


练习解答

```
File Edit Window Help Dance
% java ArrayListMagnet
zero  one  two  three
zero  one  three  four
zero  one  three  four  4.2
zero  one  three  four  4.2
```

```
import java.util.*;
public class ArrayListMagnet {
    public static void main (String[] args) {
        ArrayList<String> a = new ArrayList<String>();
        a.add(0,"zero");
        a.add(1,"one");
        a.add(2,"two");
        a.add(3,"three");
        printAL(a);
        if (a.contains("three")) {
            a.add("four");
        }
        a.remove(2);
        printAL(a);
        if (a.indexOf("four") != 4){
            a.add(4, "4.2");
        }
        printAL(a);
        if (a.contains("two")) {
            a.add("2.2");
        }
        printAL(a);
    }
    public static void printAL(ArrayList<String> al) {
        for (String element : al) {
            System.out.print(element + "  ");
        }
        System.out.println(" ");
    }
}
```

JavaCross 解答



Sharpen your pencil

写下你自己的理解，根据给出答案的每个单词的含义，写下它们简单的、难的或更技术的含义。

Across

1. ______________________
6. ______________________
7. ______________________
9. ______________________
12. ______________________
13. ______________________
14. ______________________
16. ______________________
17. ______________________
19. ______________________
21. ______________________
22. ______________________
23. ______________________
24. ______________________

Down

2. ______________________
3. ______________________
4. ______________________
5. ______________________
8. ______________________
10. ______________________
11. ______________________
15. ______________________
16. ______________________
18. ______________________
20. ______________________
21. ______________________

对象村的优质生活



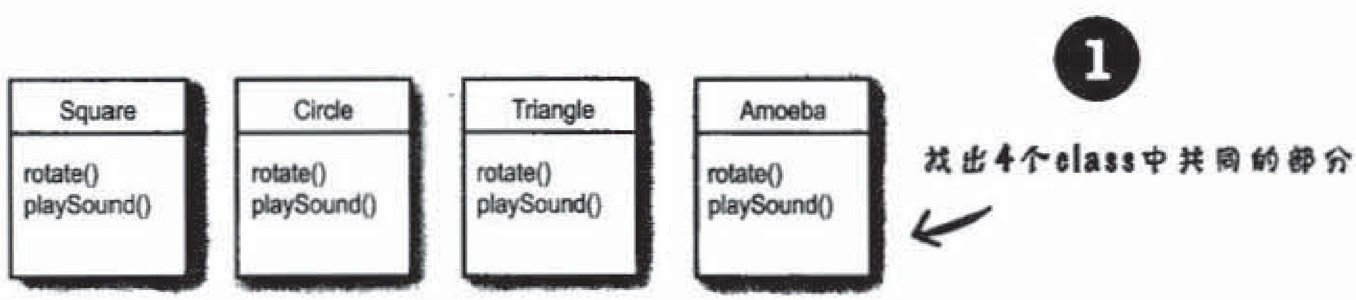
规划程序时要考虑到未来。如果有某种方法能够让你少写点Java程序、多点旅游假期，这对你会有多大的价值？如果你可以写出让他人能够很容易扩充的程序代码呢？你是否对编写出非常有适应性、可以应付最后一刻修改规格的讨厌鬼呢？如果是的话，那你今天真是幸运，因为只要花一个小时，你就可以获得这样的能力。在处理多变的计划时，你会学习到5个更好的设计步骤、3个多态的技巧以及8种让程序更有适应性的方法，此外还有4项对继承的建议。别犹豫了，有这么好的学习机会让你能够获得设计上的自由与程序的适应性，你还不赶快行动？前50名来电者现在还会送你高等抽象的概念！

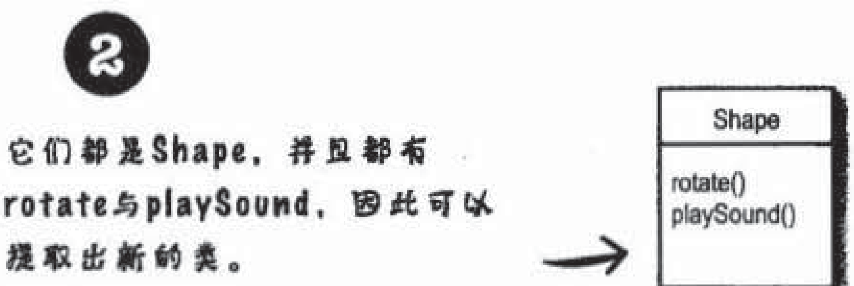
椅子大战的回顾

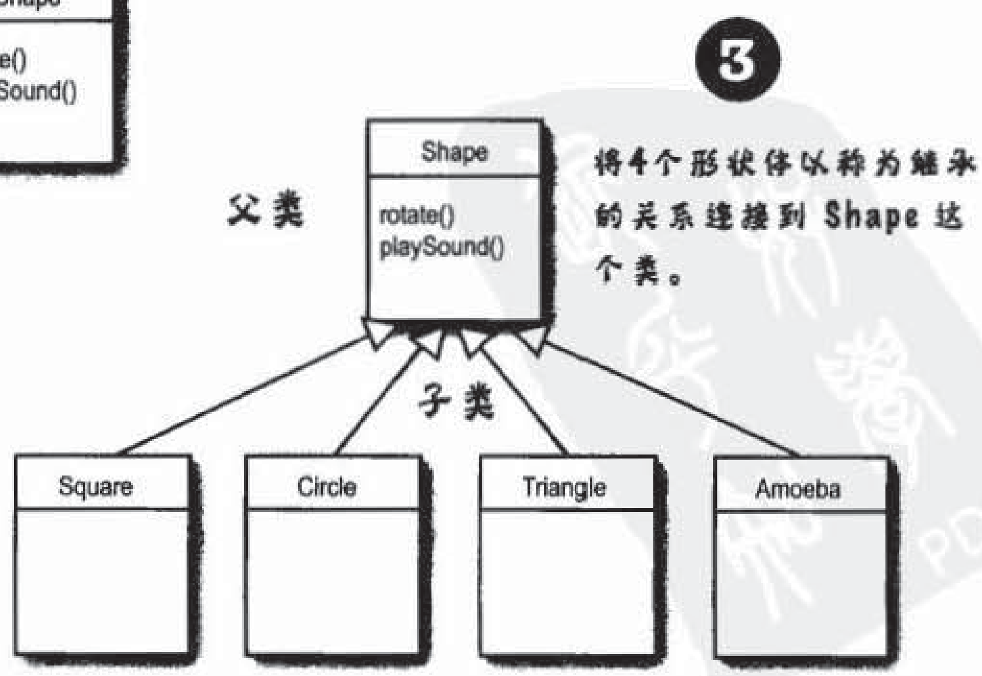
还记得第2章的宝椅争夺战吗？我们要从阿珠（面向过程派）与阿花（面向对象派）两人你死我活、惊天地泣鬼神的斗争中查看继承的基本概念。

阿珠：“你有重复的程序代码！4个Shape都有旋转的程序。这样的设计实在很蠢。如此你必须维护4个不同的rotate()方法，这档一点效率都没有”。

阿花：“我猜你一定没有看到最终的设计。阿珠，让我告诉你什么叫做继承”。







这可称为“Square继承自Shape”、“Circle继承自Shape”等。rotate()与playSound()已经从次级的类中移开。

Shape是底下4个子类的父类。次级类会继承上级类的方法。也就是说，子类会自动获得父类的功能。

那阿米巴的rotate()要怎么办？

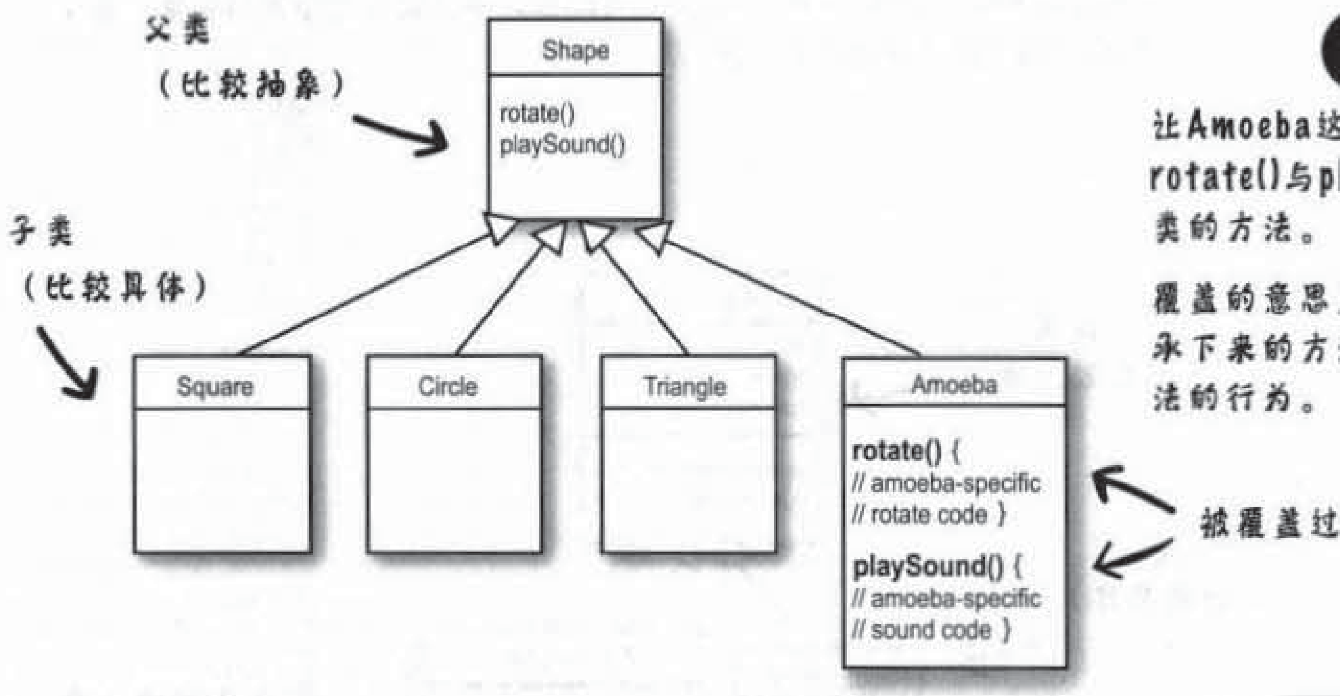
阿珠：“问题不就出在这里吗？阿米巴形状会需要完全不同的rotate与playSound程序”。



阿花：“那叫方法”。

阿珠：“如果阿米巴也是继承自Shpae，那旋转的功能不就通通一样吗？”

阿花：“问得好。Amoeba这个类可以覆盖（override）Shpae的方法。Java虚拟机会知道在遇到Amoeba时使用不同的 rotate()”。



4

让Amoeba这个类去覆盖过rotate()与playSound()这两个父类的方法。

覆盖的意思是由子类重新定义继承下来的方法以改变或延伸此方法的行为。



BRAIN POWER

你要如何用继承结构来表示家猫与老虎？被驯养的猫是一种特殊版本的老虎吗？哪一个会是子类？而哪一个才是父类呢？或两者都是某个类的子类呢？猫一天要睡几小时？方便面应该泡多久才好吃？

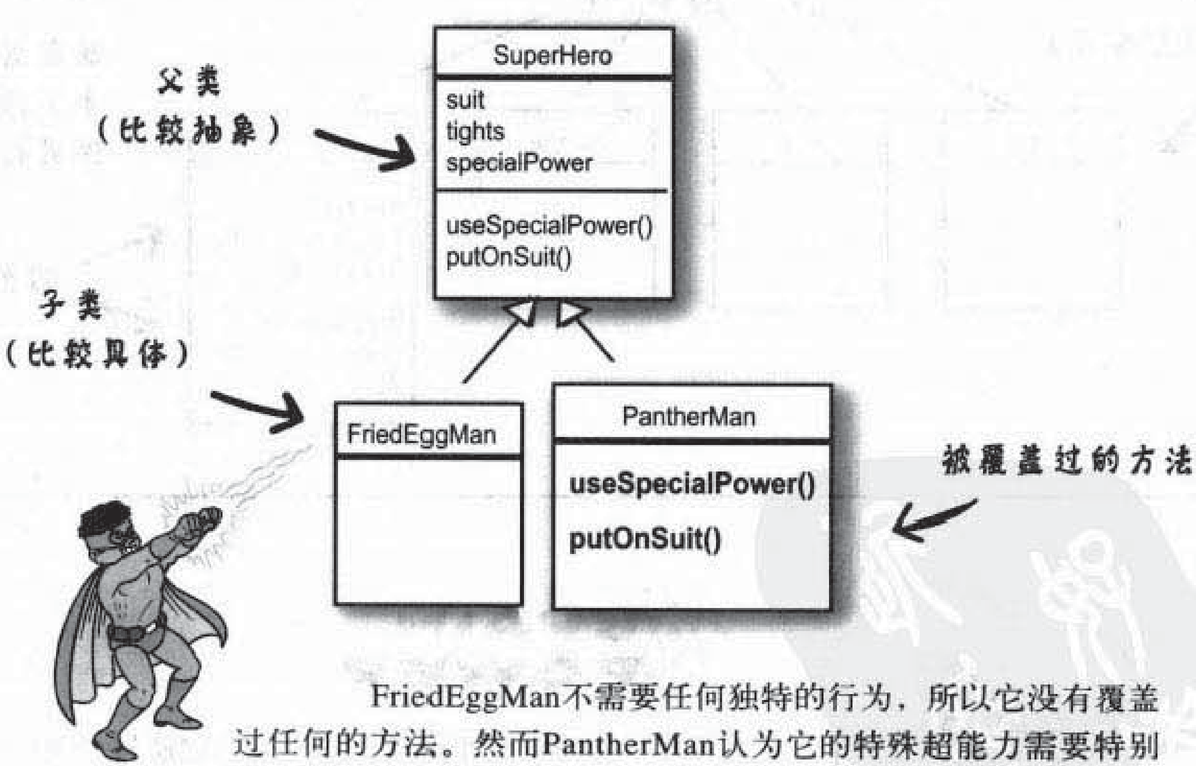
设计出继承结构之后，会有哪些方法需要被覆盖呢？翻页之前先想一想。

了解继承

在设计继承时，你会把共同的程序代码放在某个类中，然后告诉其他的类说此类是它们的父类。当某个类继承另一个类的时候，也就是子类继承自父类。

以Java的方式说，这是“子类继承父类”。继承的关系意味着子类继承了父类的方法。当我们提及“类的成员”时，成员的意思就是实例变量和方法。

举例来说，如果PantherMan是个SuperHero的子类，则PantherMan会自动地继承SuperHero的实例变量和方法，包括了suit、tights、specialPower、useSpecialPower()等。但PantherMan可以加入自己的方法和实例变量，也可以覆盖掉继承自SuperHero的方法。



FriedEggMan不需要任何独特的行为，所以它没有覆盖过任何的方法。然而PantherMan认为它的特殊超能力需要特别处理过的方法，所以就覆盖掉useSpecialPower()与putOnSuit()。

实例变量无法被覆盖掉是因为不需要，它们并没有定义特殊的行为。PantherMan可以将继承下来的tights设定成紫色，而FriedEggMan可以自行选择白色。

继承的范例

```
public class Doctor {

    boolean worksAtHospital;

    void treatPatient() {
      // 执行检查

    }
}
```

```
public class FamilyDoctor extends Doctor {

    boolean makesHouseCalls;
    void giveAdvice() {
      // 提出诊断
    }

}
```

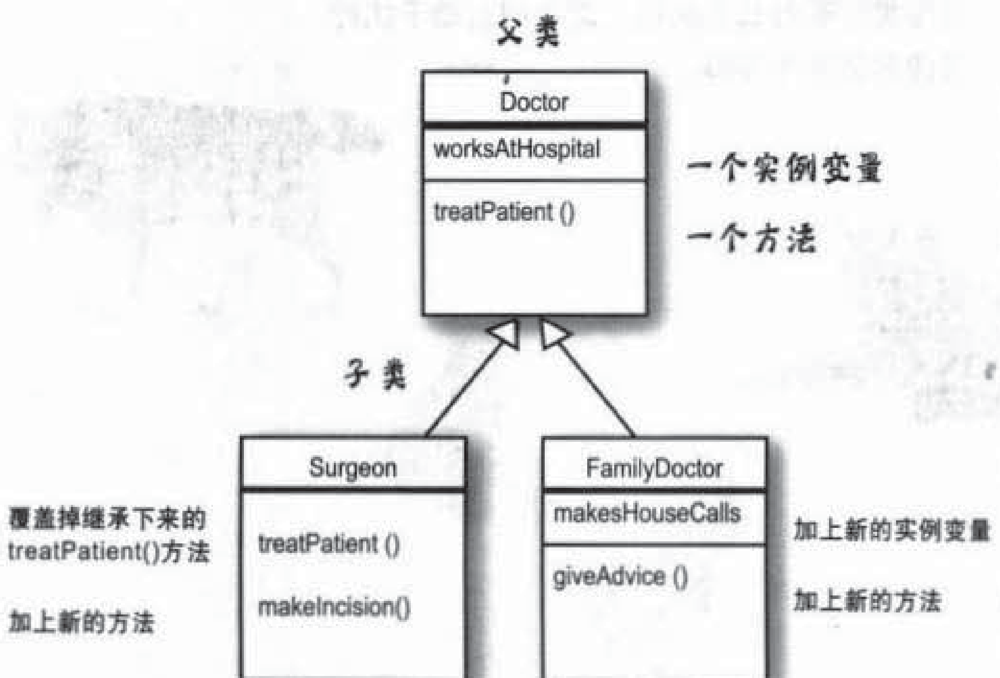
```
public class Surgeon extends Doctor{

   void treatPatient() {
      // 进行手术
   }

   void makeIncision() {
      // 截肢（好恶心！）
   }
}
```



Sharpen your pencil



Surgeon有多少个实例变量？ ____

FamilyDoctor有多少个实例变量？ ____

Doctor有多少个方法？ ____

Surgeon有多少个方法？ ____

FamilyDoctor有多少个方法？ ____

FamilyDoctor可以执行treatPatient()吗？ ____

FamilyDoctor可以执行makeIncision()吗？ ____

设计动物仿真程序的继承树

假设你要设计一个仿真系统程序，可以让用户设定将一群动物丢到某种环境中以观察会发生什么事情。现在不需要写出程序，我们只在乎设计。

我们已经被告知一部分会用到的动物，但是并不知道还有多少种动物会加进来。每个动物都会用一个对象来表示，且动物会在环境中活动，执行任何被设计出的行为。

这个程序必须能够在任何时间加入新类型的动物。

我们首先要辨别出所有动物都有的、抽象的共同特征，然后以这些共同特征设计出能够让所有动物加以扩充的类。

❶ 找出具有共同属性和行为的对象



这6种动物有什么共同点？这么问可以帮助我们执行第二个步骤。

这些类型有什么相关性？这么问有助于执行第四到第五个步骤。











用继承来防止子类中出现重复的程序代码

我们有5个实例变量：

picture：动物JPEG图像的名称。

food：此动物所吃的食物。现在只有meat和grass两种值。

hunger：代表饥饿程度的int值。它会根据动物吃了多少东西而改变。

boundaries：代表动物活动范围区域的长宽。

location：动物在活动区域中的X与Y坐标。

还有4个方法：

makeNoise()：动物发出声音的行为程序。

eat()：动物遇到食物时的行为程序。

sleep()：睡眠的行为程序。

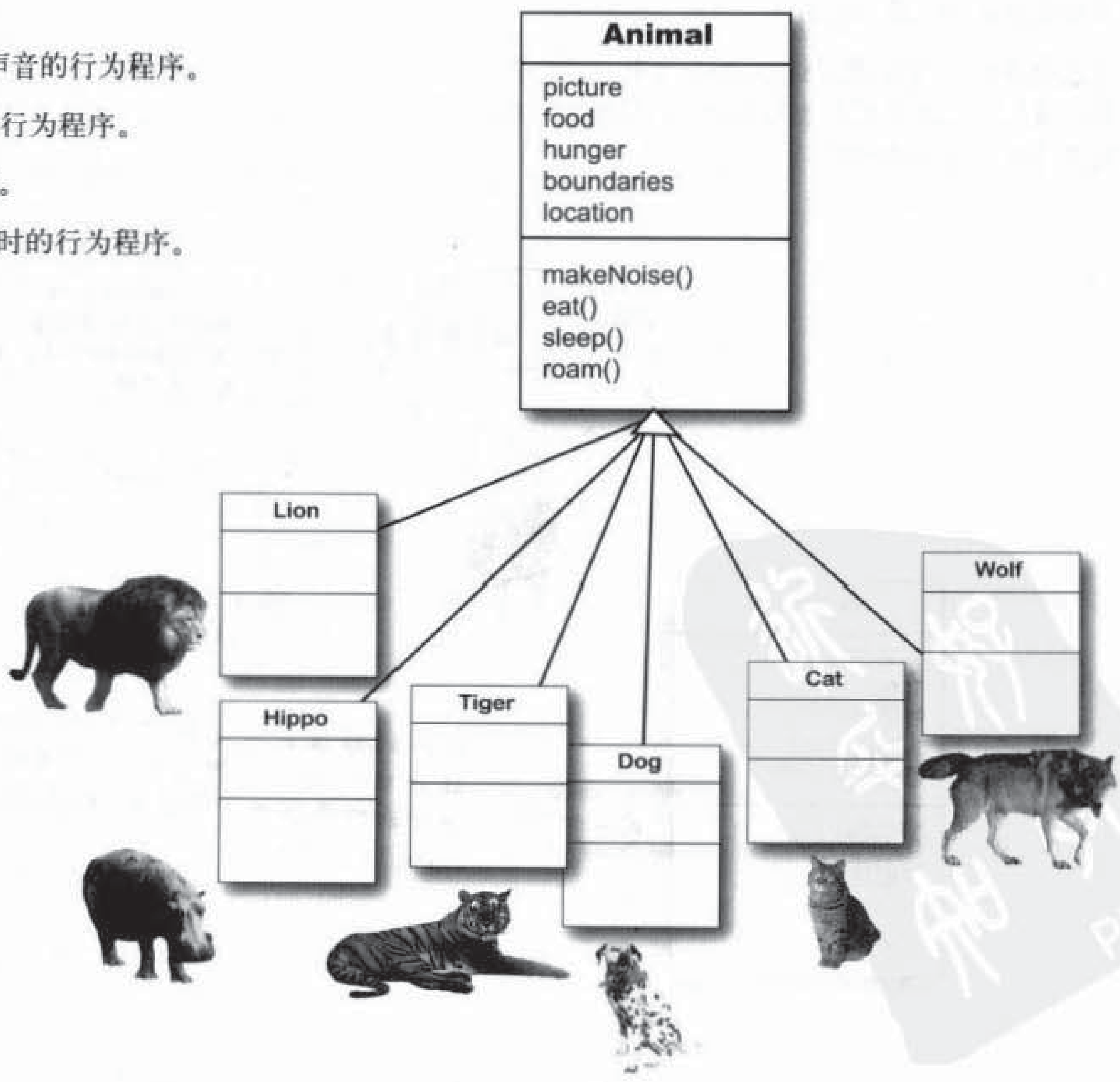
roam()：不在进食或睡眠时的行为程序。

2

设计代表共同状态与行为的类

这些对象都是动物，因此我们可以用Animal 作为共同父类的名称。

我们会把所有动物都需要的方法和实例变量加进去。



动物都以同样的方法进食吗？

假设我们都同意一件事：所有Animal类型上的实例变量都合用。狮子的picture带有它的图片路径food的值是meat。猫的图片就是猫、猫的食物是meat（其实有养猫的人都知道猫也会吃草）。所以实例变量没有问题，但是行为程序呢？

我们应该覆盖哪些方法呢？

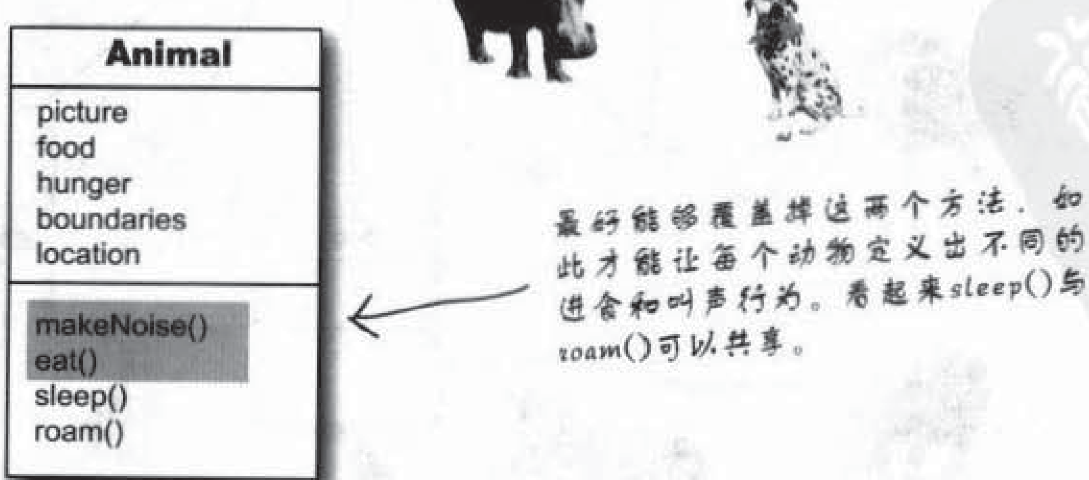
狮子的叫声会跟河马一样吗？也许你认为一样，但是我们会根据类型设计出不同的行为程序。当然，我们也可以用实例变量来存放声音文件的路径值，而让makeNoise()方法都执行相同的动作。但是有时候行为的复杂程度不只是如此而已。

因此就跟阿米巴虫覆盖过rotate()这个方法的例子一样，我们会让某些行为使用各个类自行指定的程序，而不是使用共同的程序。

3 决定子类是否需要让某项行为（也就是方法的实现）有特定不同的运作方式

观察Animal这个类之后，我们认为eat()与makeNoise()应该由各个子类自行覆盖。





寻找更多抽象化的机会

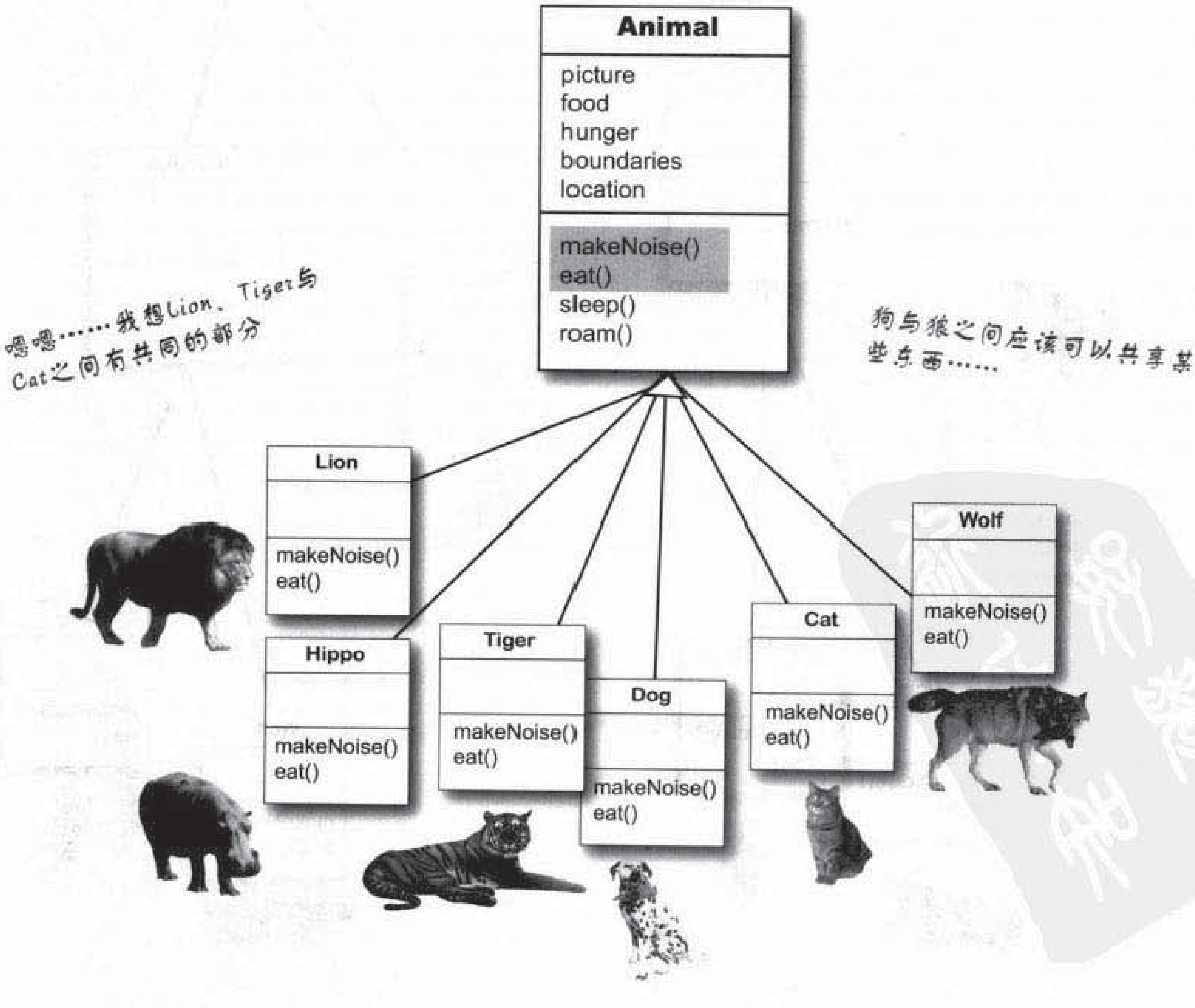
类的继承结构已经大致成型。我们让每个子类都去覆盖掉makeNoise()与eat()这两个方法，因此狗不会喵喵叫（这对双方来说都是种侮辱）、河马也不会抢狮子的食物。

但或许我们还能做更多的设计。我们必须观察Animal的子类找寻是否有可以组织归纳使用共同程序代码的部分。看起来小红帽的好朋友大野狼跟狗有共同的部分。猫、狮子与老虎也有共同的部分。

4

通过寻找使用共同行为的子类来找出更多抽象化的机会

我们观察到Wolf与Dog可能有某些共同的行为，在Lion、Tiger、Cat之间也是。



5 完成类的继承层次

因为动物本来就有组织化的层次（界、门、纲、目、科、属、种），我们可以用这些层次来制作有意义的类设计。我们使用犬科和猫科动物的分类来作出 Feline与Canine这两个类。

我们决定Canine使用共同的roam()方法，因为它们都以相同的方式移动。Feline之间也是差不多。而Hippo则持续使用继承下来的roam()方法。

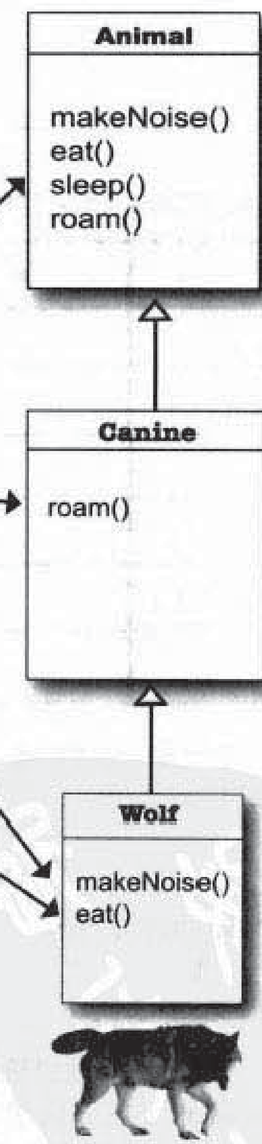
所以这个层次就这么完成了，稍后我们会再回到这个部分。



调用哪个方法？

Wolf这个类有4个方法。其中一个继承自Animal，一个来自Canine（实际上也是覆盖过Animal的方法），还有两个是自己覆盖过的。当你创建出一个Wolf对象并赋给它变量时，你可以使用圆点运算符来调用变量所引用对象的方法。但是这会调用哪个版本的方法呢？

创建大野狼对象	`Wolf w = new Wolf();`
调用大野狼的版本	`w.makeNoise();`
调用犬科的版本	`w.roam();`
调用大野狼版	`w.eat();`
调用动物版本	`w.sleep();`



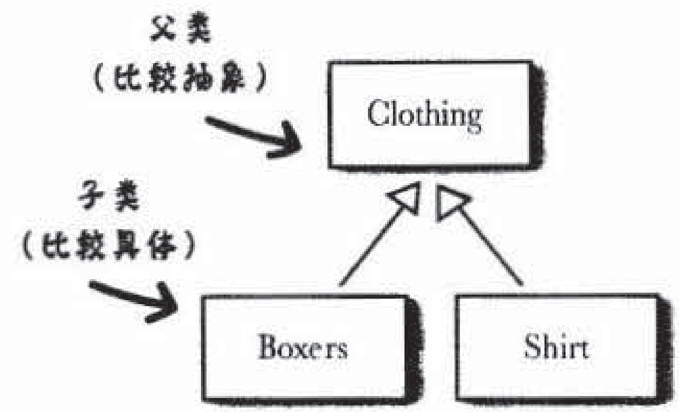
当你调用对象引用的方法时，你会调用到与该对象类型最接近的方法。换句话说，最低阶的会胜出！

“最低阶”的意思是在层次树的最下方。Canine比Animal低，而Wolf是在Canine的下方，因此Java虚拟机会从Wolf开始找起。如果Java虚拟机找不到Wolf版的方法，它会往上寻找直到找到为止。

调用哪个方法？

类	父类	子类
Clothing	---	Boxers, Shirt
Boxers	Clothing	
Shirt	Clothing	

继承表



继承类图

Shapen your pencil

在这里画出类图

找出下列类之间的关系填在空格中

类	父类	子类
Musician		
Rock Star		
Fan		
Bass Player		
Concert Pianist		

由上至下为音乐家、摇滚巨星、歌迷、Bass 手、钢琴演奏家。

there are no Dumb Questions

问： **你说Java虚拟机会从继承关系的树形图最下方开始搜索方法，要是没有找到的时候会发生什么事情？**

答： 好问题！但是你不用担心这件事。编译器会保证引用特定的方法是一定能够被调用到，但在执行期它不会在乎该方法实际上是从哪个类找到的。以Wolf为例，编译器会检查sleep()这个方法，但却不管sleep()实际上是定义在Animal这个类。要记得如果某个类继承了一个方法，它就会有那个方法。方法在哪里定义对于编译器来说不重要。但在执行期，Java虚拟机就是有办法找到正确的。这个正确的意思是最接近该类型的版本。

"是一个"与"有一个"

当一个类继承自另外一个类时，我们会说这是子类去继承父类。若你想要知道某物是否应该要继承另一物时，则可以用 IS-A 测试来检验。

三角形是一个多边型……嗯，没错。

外科医生是一个医生……OK。

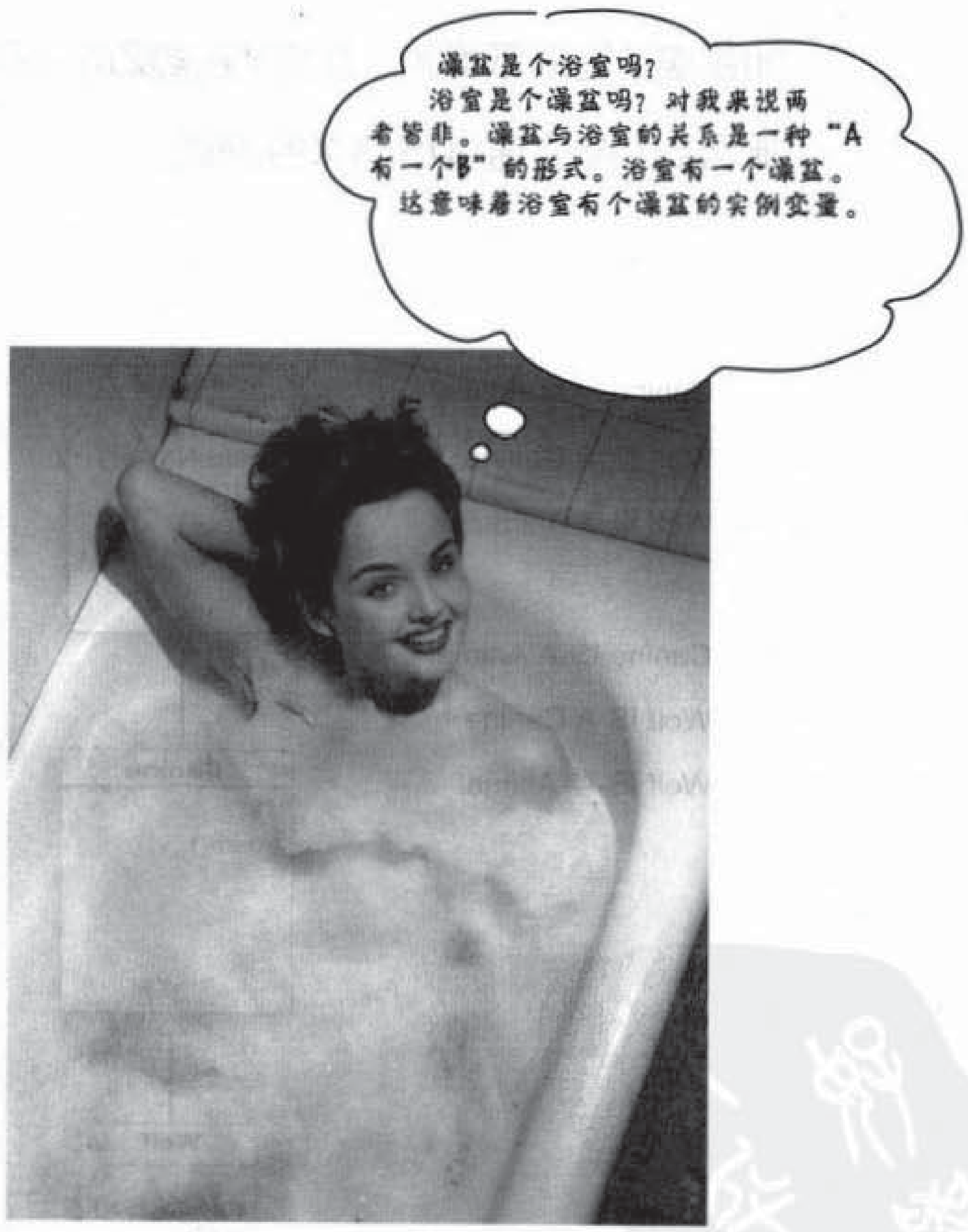
哈啰凯蒂是一个猫……算是吧。

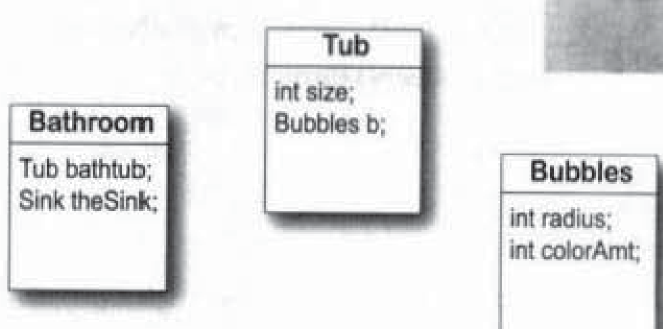
澡盆是一个浴室……失败！

大肠没有洗干净……失败中的失败！

要确认你的设计是否正确，使用这样的测试来加以检验。如果不合理，表示你的设计有问题。

浴室与澡盆确实有关联，但不是继承上的关系。浴室与澡盆发生的是HAS-A的关系。如果"浴室有一个澡盆"成立的话，这表示浴室带有澡盆的实例变量。也就是说浴室会有个澡盆的引用，但是浴室并没有继承过澡盆。





浴室有一个澡盆且澡盆有一个泡泡。

但这里没有继承关系。

等一下！还有！

IS-A测试适用在继承层次的任何地方。如果你的继承层次树设计得很好，那么所有的子类都应该通过任一个上层父类的IS-A测试。

如果类Y是继承类X，且类Y是类Z的父类，那么Z应该能通过IS-A X的测试。

Canine 继承 Animal

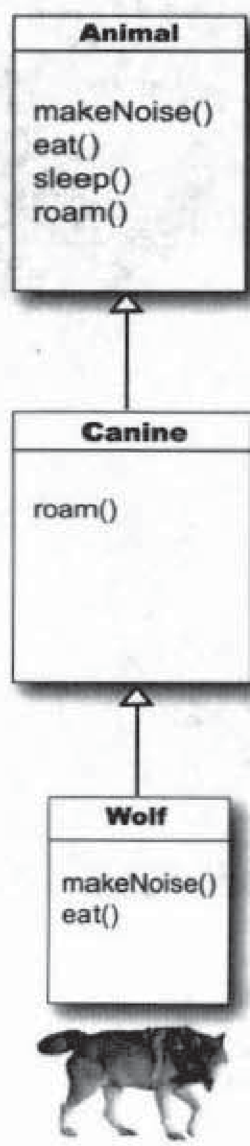
Wolf 继承 Canine

Wolf 继承 Animal

Canine IS-A Animal

Wolf IS-A Canine

Wolf IS-A Animal



就像此处所显示出的继承树，你一定可以说："Wolf继承Animal"或"Wolf IS-A Animal"。只要Animal位于Wolf之上，Wolf IS-A Animal就一定会成立。

这张Animal继承图说明了：

"Wolf是一个Canine，因此Canine能做的事情Wolf都能做，而Wolf也是个Animal，所以Animal能做的事情Wolf也都能做"。

就算Wolf已经覆盖掉某些来自Animal或Canine的方法也一样。对其他的程序来说，它们只要知道Wolf能够执行这4个方法就行。至于它是怎么做的，或者它是覆盖过哪个类的，则一点都不重要。至少有一件事情可以确定，Wolf一定可以makeNoise()、eat()、sleep()和roam()。

如何知道继承设计是对的？

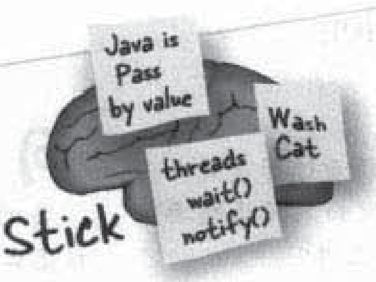
很明显，事情不只是这样而已，但是我们会在下一章讨论更多有关面向对象的概念（最终我们还是得改善这一章所作出的设计）。

虽然如此，我们还是要提出IS-A测试这个建议。如果“X IS-A Y”合理，则这两者或许存在于同一个继承结构下。也有可能两者根本是相同的，或者刚好有相同的行为。

注意：继承概念下的IS-A是个单向的关系！

“三角形是一个多边形”这是合理的，所以你可以从多边形中扩充出三角形。

但是反过来说“多边形是一个三角形”并不合理，所以多边形并不是从三角形中extend出来的。要记得X IS-A Y隐喻着 X 可以做出任何Y可以做的事情（且还可能会做出更多的行为）。



Make it Stick

玫瑰是红的，

球不是这么踢的，

啤酒是一种饮料，饮料不一定是啤酒。

好了，算你作吟诗作对，但千万别把你的诗拿去投稿！

Sharpen your pencil

在合理的叙述旁边打勾

- [ ] Oven extends Kitchen
- [ ] Guitar extends Instrument
- [ ] Person extends Employee
- [ ] Ferrari extends Engine
- [ ] FriedEgg extends Food
- [ ] Beagle extends Pet
- [ ] Container extends Jar
- [ ] Metal extends Titanium
- [ ] GratefulDead extends Band
- [ ] Blonde extends Smart
- [ ] Beverage extends Martini

提示：运用IS-A测试。

there are no Dumb Questions

问： 我们已经看过子类是如何继承父类的方法，但如果父类想要使用子类的方法时该怎么办?

答： 父类不一定要知道它的子类。或许有人会写出一个类并且在多年之后被他人扩充过。回到问题，就算父类与子类是同一个人编写的，继承的方向也不可能反过来。你可以想象小孩继承了父母的遗传，而遗传不会有别的方向。

问： 如果在子类中还打算引用父类的方法然后再加上额外的行为应该怎么办?

答： 这是可行的！事实上这个功能非常重要。扩充本来就有扩充和延伸的意思。

```
public void roam() {
   super.roam();
     // my own roam stuff
}
```

这会先执行super版的 roam()然后再回来执行sub版自定义的行为或功能

你可以在父类中设计出所有子类都适用的功能实现。让子类可以不用完全覆盖掉父类的功能，只是再加上额外的行为。你可以通过super这个关键词来取用父类。

谁开保时捷？又有谁开保肝丸？

（如何知道子类能够继承下来哪些东西？）



子类可以继承父类的成员。这包括了实例变量和方法，然而稍后我们才会提到其他会被继承的东西。父类可以通过存取权限决定子类是否能够继承某些特定的成员。

在这本书中，我们会讨论4种存取权限。下面列出这4种权限，左边是最受限制的，而越往右边限制程度越小：

private	default	protected	public

存取权限（access level）控制了谁可以接触什么，这对编写出坚固、设计良好的Java程序来说是很重要的。现在我们先看看public与private两项：

public类型的成员会被继承

private类型的成员不会被继承

当子类把成员继承下来时会把它们当作是自己定义的一样。例如说当某个形状体继承 Shape 时，就会有rotate()与 playSound()这两个方法。

任一类的成员包含有自己定义出的变量和方法再加上从父类所继承下来的任何东西。

注意：在附录B和第16章中会看到更多关于default和protected的信息。

你会善用继承吗？你滥用继承了吗？

虽然下面要讨论的规则有些原因是我们现在不会先说明的，但是记住这些规则能够帮助你进行更好的继承设计，这些背后的因素会在后面的章节再加以介绍。

当某个类会比其父类更具有特定意义时使用继承。例如说美国短毛猫是一种特定品种的猫，所以从猫中扩充出美国短毛猫是很合理的。

在行为程序（实现程序代码）应该被多个相同基本类型类所共享时，应该要考虑使用继承。举例来说，方形、圆形、三角形都需要旋转和播放声音，因此将这些功能放在它们的父类上面是很合理的，并且这样也比较好维护和扩充。然而，要注意到虽然继承是面向对象程序设计的一项关键特征，但却不一定是达成重用行为程序的最佳方式。我们会教你如何运用继承，这通常也是不错的选择，但有时常用的“设计模式（design pattern）”也会提出更微妙且更有适应性的选择。如果你不太清楚设计模式是什么，在你能够掌握面向对象程序设计的概念之后，不妨看看《Head First设计模式》，这本书也有中译本。

若两者间的关系对于继承结构来说并不合理，则不要只是因为打算要重用其他类的程序代码而运用继承。例如，在设计钢琴对象时，不能因为想要借用河马对象的发声程序就让这两个八竿子打不着的对象产生继承上的关系。这完全不合理！（应该要创建出发音对象，然后让钢琴与河马都用HAS-A关系来运用此对象才对）。

如果两者间不能通过IS-A测试就不要应用继承关系。一定要确定子类是父类一种更特定的类型才可以。

要点

- 子类是extends父类出来的。
- 子类会继承父类所有public类型的实例变量和方法，但不会继承父类所有private类型的变量和方法。
- 继承下来的方法可以被覆盖掉，但实例变量不能被覆盖掉。
- 使用IS-A测试来验证继承结构的合理性。
- IS-A关系是单方向的，河马是动物，但动物不一定是河马。
- 当某个方法在子类中被覆盖过，调用这个方法时会调用到覆盖过的版本。
- 如果类Y是extends类X，且类Y是类Z的父类，则Z应该能通过IS-A X的测试。

继承到底有什么意义？

通过设计继承的过程你可以累积面向对象的经验值。通过提取出一组类间共同的抽象性，你能够排除掉重复的程序代码而将这个部分放在父类中。如此一来，如果有共同的部分需要改变，就只会有一个地方要修改而已，且这样的改变会应用到所有继承此行为的类。修改之后只需要重新编译就行，不必动子类！

换上改变过的父类，则所有扩充过它的类都会自动使用到新的版本。

Java程序只是由一堆类组成的，因此，子类不需要重新编译就能运用到新版本的父类。如果父类没有破坏到子类，万事都会OK（稍后我们会讨论到何谓破坏，现在你可以先想象如果父类修改了方法的参数数目、类型或返回类型会怎么样）。

① 避免了重复的程序代码

在单一的位置定义共同程序代码，然后让子类继承父类的程序代码。当你想要改变这个行为程序时，只需修改这个地方，而子类就会发生同样的改变。

② 定义出共同的协议



继承让你可以确保某个父型之下的所有类都会有父型所持有的全部方法*

也就是说，你会通过继承来定义相关类间的共同协议

当你在父类中定义方法时，它们会被子类继承，这样你就是在对其他程序代码声明："我所有的子类（例如说subclass）都能用这些方法来执行这几项工作……"。

也就是说你拟出了一份"合约"。

Animal这个类拟出所有动物子型的共同协议：



要记得，当我们说所有的动物时，意思是Animal以及所有继承过Animal的类。也就是说在继承层次上方有Animal的任何一个类。

不过我们还没有讨论到最精采的部分——多态（polymorphism），这要留到最后。

注意下面这段说明，我们会持续在接下来的几页中解释它的意义：

当你定义出一组类的父型时，你可以用子型的任何类来填补任何需要或期待父型的位置。

这段说明很重要……

因为你会体会到多态的好处。

因为我会……

通过声明为父型类型的对象引用来引用它的子型对象。

对我来说……

这是编写出真正具有适应性的程序代码的机会。程序会变得更简洁、更有效率、更简单。程序不但容易开发而且也更容易扩展。

这样你就可以在同事更新程序的同时好好地度个假，他们甚至不需要你的源代码或打扰你就可以继续工作。

你会在下一页看到它是如何运行的。

我们其实也不认识你，但是个人相信上面这样的说法蛮吸引人的。



*"全部方法"的意思是"全部可继承的方法"。更准确的说法是所有public类型的方法，稍后我们还会对这个定义作些微小的更正。

若要观察多态是如何运行的，我们就必须先退回去看一般声明引用和创建对象的方法……

对象声明、创建与赋值的3个步骤：

```
      1           3      2
Dog myDog  =  new Dog();
```

❶ 声明一个引用变量

```
Dog myDog = new Dog();
```

要求Java虚拟机分配空间给引用变量，并将此变量命名为myDog。此引用变量将永远被固定为Dog类型。



❷ 创建对象

```
Dog myDog = new Dog();
```

要求Java虚拟机分配堆空间给新建立的Dog对象。



❸ 连接对象和引用

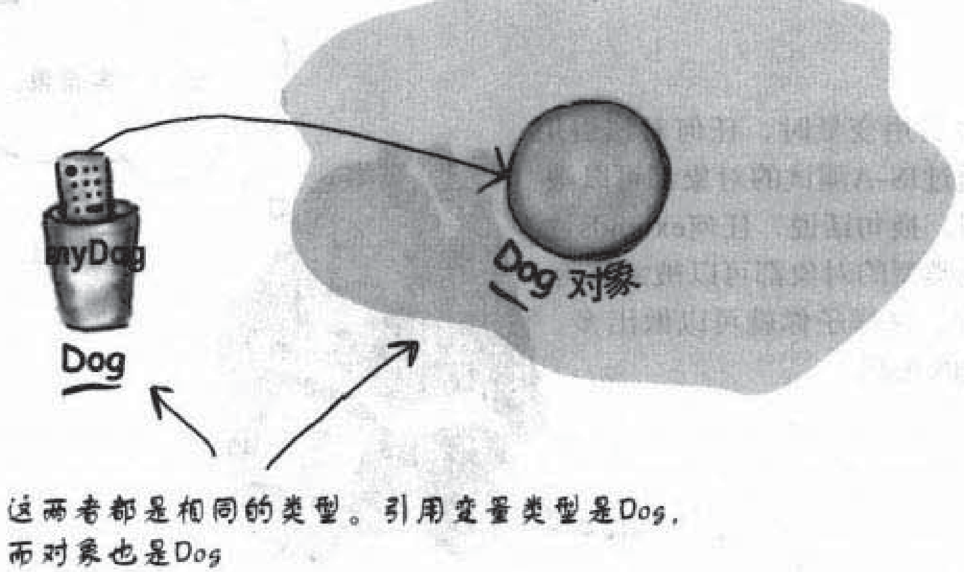
```
Dog myDog = new Dog();
```

将新的Dog赋值给myDog这个引用变量。换言之就是设定遥控器。



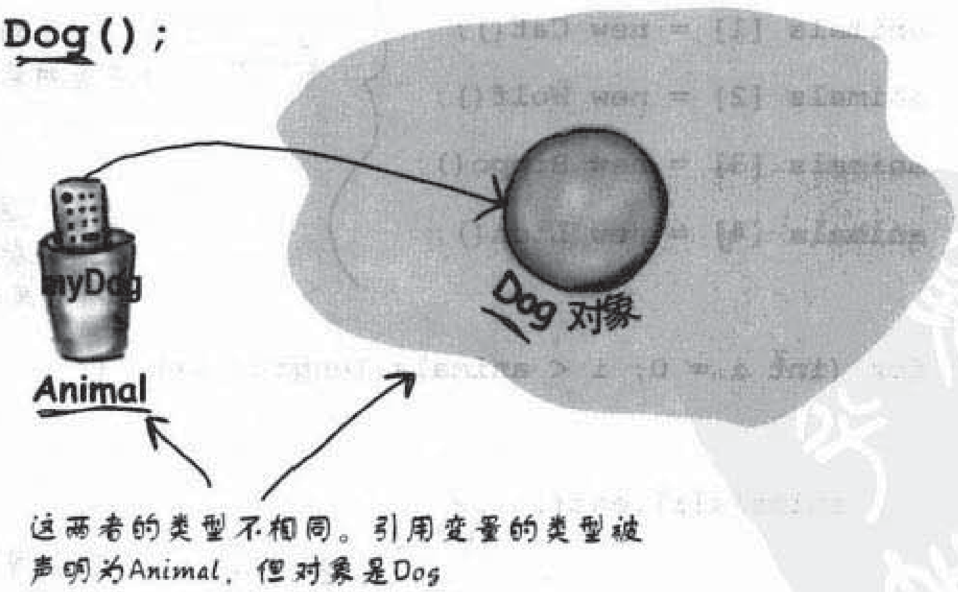
重点在于引用类型与对象的类型必须相符。

在此例中，两者皆为Dog。



但在多态下，引用与对象可以是不同的类型。

```
Animal myDog = new Dog();
```



运用多态时，引用类型可以是实际对象类型的父类

当你声明一个引用变量时，任何对该引用变量类型可通过IS-A测试的对象都可以被赋值给该引用。换句话说，任何extends过声明引用变量类型的对象都可以被赋值给这个引用变量。这样子你就可以做出多态数组这一类的东西。



OK，或许举个例子会比较清楚。

```
Animal[] animals = new Animal[5];
animals [0] = new Dog();
animals [1] = new Cat();
animals [2] = new Wolf();
animals [3] = new Hippo();
animals [4] = new Lion();

for (int i = 0; i < animals.length; i++) {

    animals[i].eat();

    animals[i].roam();

}
```

声明Animal类型的数组。也就是说一个会保存Animal类型对象的数组

但是注意到这边……你可以放任何Animal的子类对象进去

这就是多态最强的地方，你可以将数组的元素逐个调出来当作是Animal来操作

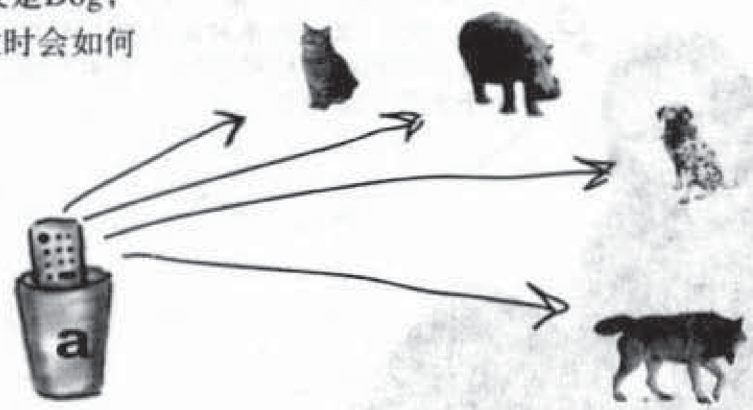
当i为0的时候，这会调用Dog的eat()

当i为1的时候，这会调用Cat的roam()

不仅如此！还有呢！

参数和返回类型也可以多态！

如果你声明一个父类的引用变量，比如说Animal，并赋子类对象给它，假设是Dog，想象一下此变量被当作方法的参数时会如何运作……



```
class Vet {

   public void giveShot(Animal a) {

     // do horrible things to the Animal at

     // the other end of the 'a' parameter

     a.makeNoise();

   }

}
```

a 参数可以用任何Animal的类型对象来当传入。执行到makeNoise()的时候，不管它引用的对象到底是什么，该对象都会执行makeNoise()

```
class PetOwner {

   public void start() {

      Vet v = new Vet();

      Dog d = new Dog();

      Hippo h = new Hippo();

      v.giveShot(d);

      v.giveShot(h);

   }

}
```

giveShot这个方法可以取用任何一种Animal。只要所传入的是Animal的子类它都能执行

会执行Dog的makeNoise()

会执行Hippo的makeNoise()



通过多态，你就可以编写出引进新型子类时也不必修改的程序。

还记得Vet这个类吗？如果你使用Animal类型来声明它的参数，则程序代码就可以处理所有Animal的子类。这意味着其他人只要注意到要扩充过Animal就可以利用你的Vet。

BRAIN POWER

为什么多态保证这么做是没有问题的？为什么我们可以放心地假设所有的子类都会跟父类一样的 方法可以通过圆点运算符调用？

there are no Dumb Questions

问： 设计子类是否有层次上的限制？最多能够设计几层？

答： 如果你观察Java API，你会看到大多数的继承的层次有深度，但是不会很深。大部分不会超过一或两层。但是也有例外，特别是在GUI类这边。之后你就会了解到保持继承树层次少一点是合理的，但实际上并没有严格层数规定（至少你应该不会碰到）。

问： 啊，我刚刚想到一件事情，如果你没有办法看到类的源程序代码，但又想要改变该类的方法，是否可以用子类的方式来做呢？用你自己好的代码设计不好的类并覆盖掉它们的方法？

答： 可以。这是面向对象一项很了不起的特征。有时这样也可以帮你省下全部重写代码的时间。

问： 你能够继承任何一个类吗？就像类的成员一样如果类是私有的你就不能继承？

答： 内部类我们还没有介绍到。除了内部类之外，并没有私有类这样的概念。但是有三种方法可以防止某个类被作出子类。

第一种是存取控制。就算类不能标记为私有，但它还是可以不标记公有。非公有的类只能被同一个包的类作出子类。

第二种是使用final这个修饰符（modifier）。这表示它是继承树的末端，不能被继承。

第三种是让类只拥有private的构造程序（constructor，第9章会说明）。

问： 你为什么会作出标识final的类？这样有什么好处？

答： 一般来说，你不会标识出final。但如果你需要安全——确保方法都会是你写的版本，此时就需要final。

问： 可不可以只用final去标识方法而不使用整个类。

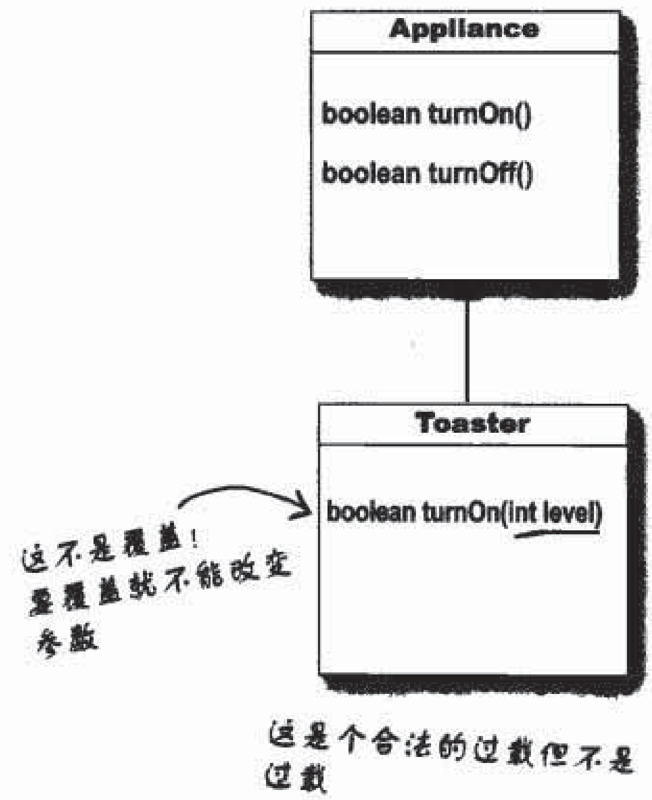
答： 如果你想要防止特定的方法被覆盖，可以将该方法标识上final这个修饰符。将整个类标识成final表示没有任何的方法可以被覆盖。

遵守合约：覆盖的规则

当你要覆盖父类的方法时，你就得同意要履约。比如，这个合约表示“我没有参数且返回布尔值”。因此，你所覆盖的方法就必须没有参数且返回布尔值。

方法就是合约的标志。

如果多态运行无误的话，Toaster版覆盖Applicance的方法就会在运行期运行。记住，编译器会寻找引用类型来决定你是否可以调用该引用的特定方法。但在执行期，Java虚拟机寻找的并不是引用所指的类型，而是在堆上的对象。因此若编译器已经同意这个调用，则唯一能够通过的方法是覆盖的方法也有相同的参数和返回类型。不然的话，就算Toaster有个取用int版本的turnOn()，还是会以Appliance引用来调用没有参数的版本。到底运行期会调用哪个版本？答案是Appliance的那一个版本。换句话说，在Toaster中的turnOn(int level) 这个方法并没有覆盖掉Appliance 的版本！



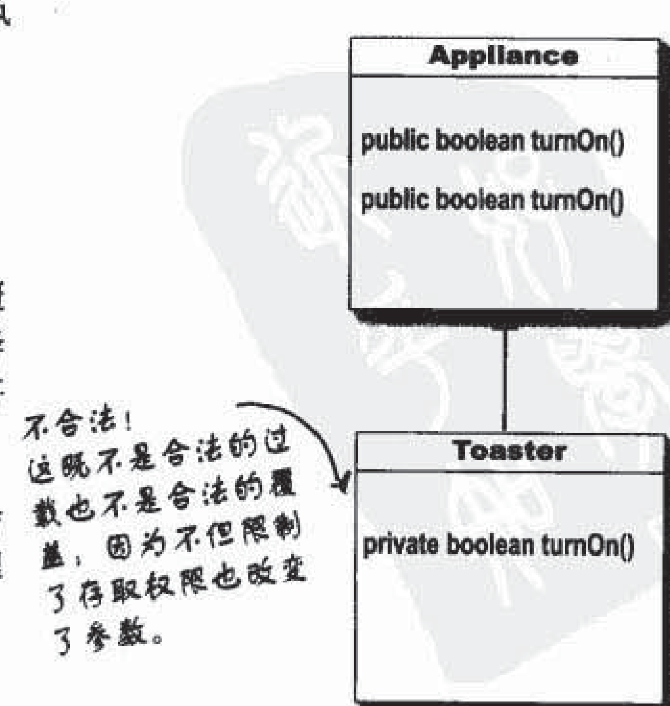
参数必须要一样，且返回类型必须要兼容

父类的合约定义出其他的程序代码要如何来使用方法。不管父类使用了哪种参数，覆盖此方法的子类也一定要使用相同的参数。而不论父类声明的返回类型是什么，子类必须要声明返回一样的类型或该类型的子类。要记得，子类对象得保证能够执行父类的一切。

不能降低方法的存取权限

这代表存取权必须相同，或者更为开放。举例来说，你不能覆盖掉一个公有的方法并将它标记为私有。这会让它以为在编译期通过的是个公有，然后突然在执行期才被Java虚拟机阻止存取。

目前为止我们看过了私有与公有这两种存取权限。另外两种会在讨论布署的章节和附录B中说明。还有关于覆盖的异常处理会在讨论异常的章节说明。



方法的重载（overload）

重载的意义是两个方法的名称相同，但参数不同。所以，重载与多态毫无关系。

重载可以有同一方法的多个不同参数版本以方便调用。比如，如果某个方法需要int，调用方就得将 double转换成int然后才能调用。若你有个重载版本取用double参数，则这样对调用方来说就简单多了。这在讨论对象生命周期的章节中关于构造函数一节会有更多的说明。

因为重载方法不是用来满足定义在父类的多态合约，所以重载的方法比较有扩展性。

重载版的方法只是刚好有相同名字的不同方法，它与继承或多态无关。重载的方法与覆盖方法不一样。

返回类型可以不同

你可以任意地改变重载方法的返回类型，只要所有的覆盖使用不同的参数即可。

不能只改变返回类型

如果只有返回类型不同，但参数一样，这是不允许的。编译器不会让这样的事情过关。就算是重载，也要让返回类型是父类版返回类型的子类。重载的条件是要使用不同的参数，此时返回类型可以自由地定义。

可以更改存取权限

你可以任意地设定overload版method的存取权限。

重载的合法范例

```
public class Overloads {

   String uniqueID;

   public int addNums(int a, int b) {
      return a + b;
   }

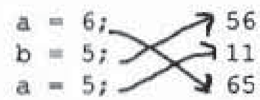
   public double addNums(double a, double b) {
       return a + b;
   }

   public void setUniqueID(String theID) {
      // lots of validation code, and then:
      uniqueID = theID;
   }

   public void setUniqueID(int ssNumber) {
      String numString = "" + ssNumber;
      setUniqueID(numString);
   }
}
```



连连看



下面有一个Java小程序。其中有一段程序不见了。你的任务是找出下面所列出的程序段与相应的输出。

并非所有的输出都有可对应的程序段，且某些输出可能会被使用多次。画条线将相符的两者连接起来。

the program:

```
class A {
  int ivar = 7;
  void m1() {
    System.out.print("A's m1, ");
  }
  void m2() {
    System.out.print("A's m2, ");
  }
  void m3() {
    System.out.print("A's m3, ");
  }
}

class B extends A {
  void  m1() {
    System.out.print("B's m1, ");
  }
}
```

```
class C extends B {
  void m3() {
    System.out.print("C's m3, "+(ivar + 6));
  }
}

public class Mixed2 {
  public static void main(String [] args) {
    A a = new A();
    B b = new B();
    C c = new C();
    A a2 = new C();
    [          ]   <- 少了3行程序代码

  }
}
```

程序候选码：

```
b.m1();
c.m2();
a.m3();
```

```
c.m1();
c.m2();
c.m3();
```

```
a.m1();
b.m2();
c.m3();
```

```
a2.m1();
a2.m2();
a2.m3();
```

输出：

```
A's m1, A's m2, C's m3, 6

B's m1, A's m2, A's m3,

A's m1, B's m2, A's m3,

B's m1, A's m2, C's m3, 13

B's m1, C's m2, A's m3,

B's m1, A's m2, C's m3, 6

A's m1, A's m2, C's m3, 13
```





我是编译器

哪一组A-B两段程序代码插入左边的类中可以通过编译并执行出下方的输出？（A插入类Monster中，B插入类Vampire中）

```
public class MonsterTestDrive {
  public static void main(String [] args) {
    Monster [] ma = new Monster[3];
    ma[0] = new Vampire();
    ma[1] = new Dragon();
    ma[2] = new Monster();
    for(int x = 0; x < 3; x++) {
      ma[x].frighten(x);
    }
  }
}

class Monster {

  A

}

class Vampire extends Monster {

  B

}

class Dragon extends Monster {
  boolean frighten(int degree) {
    System.out.println("breath fire");
    return true;
  }
}
```

```
File Edit Window Help SaveYourself
% java MonsterTestDrive
a bite?
breath fire
arrrgh
```

1

A
```
boolean frighten(int d) {
  System.out.println("arrrgh");
  return true;
}
```

B
```
boolean frighten(int x) {
  System.out.println("a bite?");
  return false;
}
```

2

A
```
boolean frighten(int x) {
  System.out.println("arrrgh");
  return true;
}
```

B
```
int frighten(int f) {
  System.out.println("a bite?");
  return 1;
}
```

3

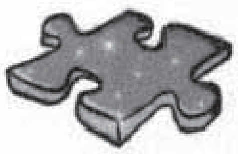
A
```
boolean frighten(int x) {
  System.out.println("arrrgh");
  return false;
}
```

B
```
boolean scare(int x) {
  System.out.println("a bite?");
  return true;
}
```

4

A
```
boolean frighten(int z) {
  System.out.println("arrrgh");
  return true;
}
```

B
```
boolean frighten(byte b) {
  System.out.println("a bite?");
  return true;
}
```





泳池迷宫

你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段可以重复使用，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

```
public class Rowboat ________ ________ {
   public ________ rowTheBoat() {
      System.out.print("stroke natasha");
   }
}
```

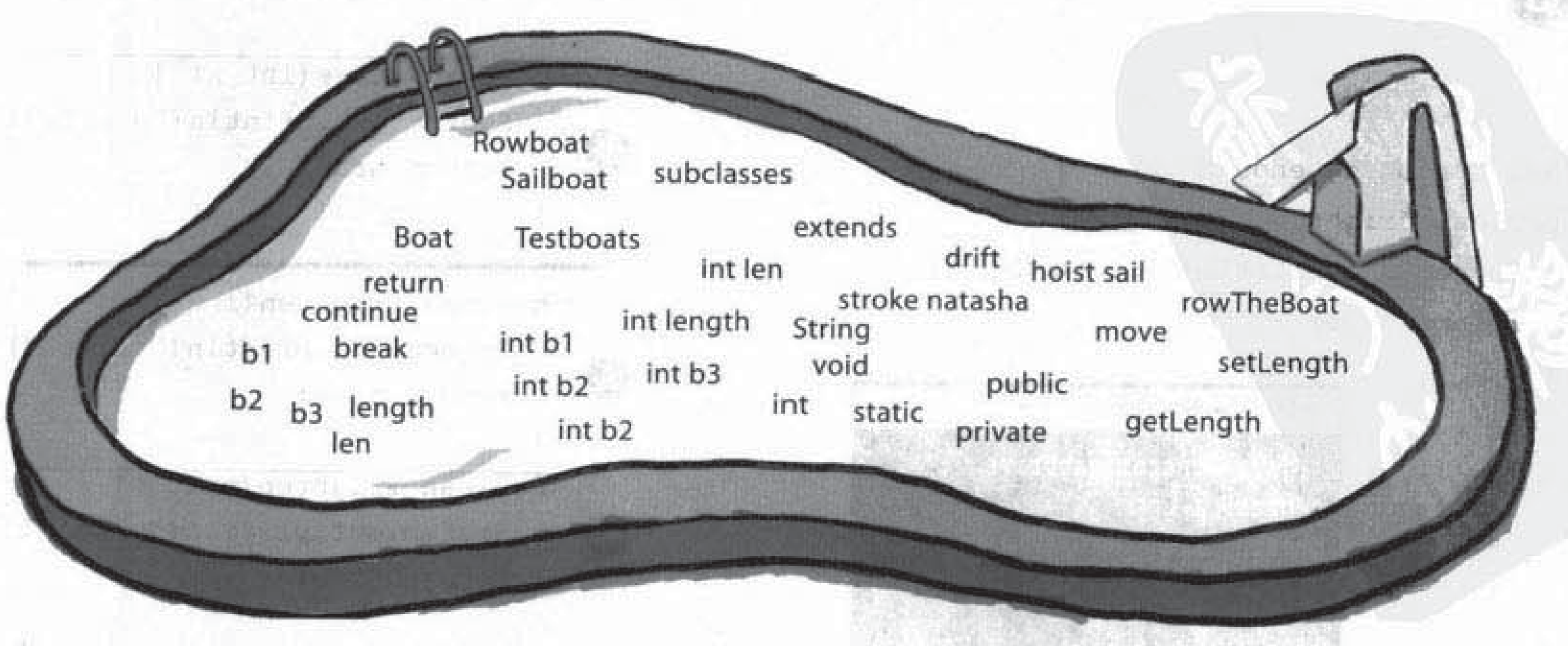
```
public class ________ {
   private int ________ ;
   ________ void ________ ( ________ ) {
      length = len;
   }
   public int getLength() {
      ________ ________ ;
   }
   public ________ move() {
      System.out.print("________");
   }
}
```

```
public class TestBoats {
   ______ ______ ______ main(String[] args){
      ________ b1 = new Boat();
      Sailboat b2 = new ________();
      Rowboat ________ = new Rowboat();
      b2.setLength(32);
      b1.________();
      b3.________();
      ______.move();
   }
}
```

```
public class ________ ________ Boat {
   public ______ ________() {
      System.out.print("________");
   }
}
```

输出：

```
drift  drift  hoist sail
```





我是编译器



第一组可以。

第二组无法通过编译，因为 Vampire 返回的类型是 int。

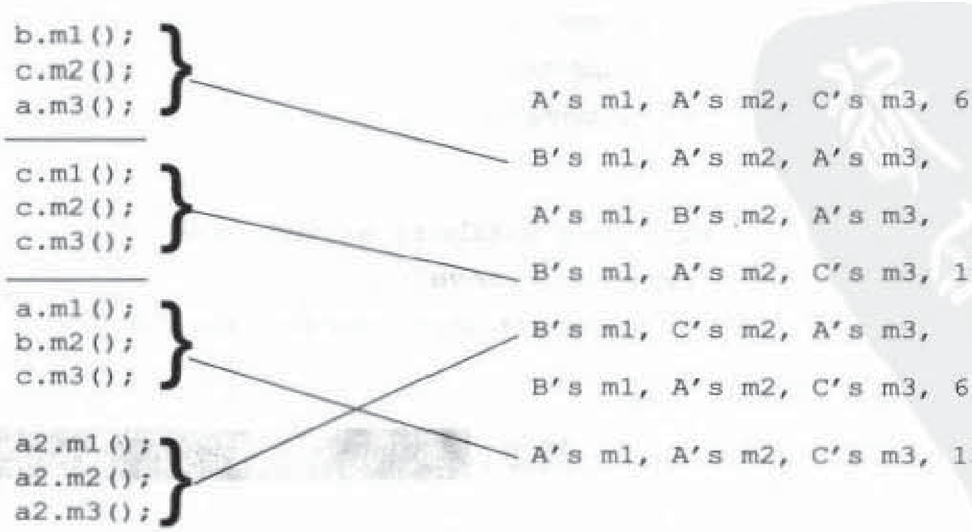
第三组与第四组可以编译，但是会产生下面这样的输出：

arrrgh

breath fire

arrrgh

连连看



```
public class Rowboat extends Boat {
   public void rowTheBoat() {
      System.out.print("stroke natasha");
   }
}
public class Boat {
   private int length ;
   public void setLength ( int len ) {
      length = len;
   }
   public int getLength() {
      return length ;
   }
   public void move() {
      System.out.print("drift  ");
   }
}
```

```
public class TestBoats {
   public static void main(String[] args){
      Boat b1 = new Boat();
      Sailboat b2 = new Sailboat();
      Rowboat b3 = new Rowboat();
      b2.setLength(32);
      b1.move();
      b3.move();
      b2.move();
   }
}
public class Sailboat extends Boat {
   public void move() {
      System.out.print("hoist sail ");
   }
}
```

输出： drift drift hoist sail

PDG

8 接口与抽象类

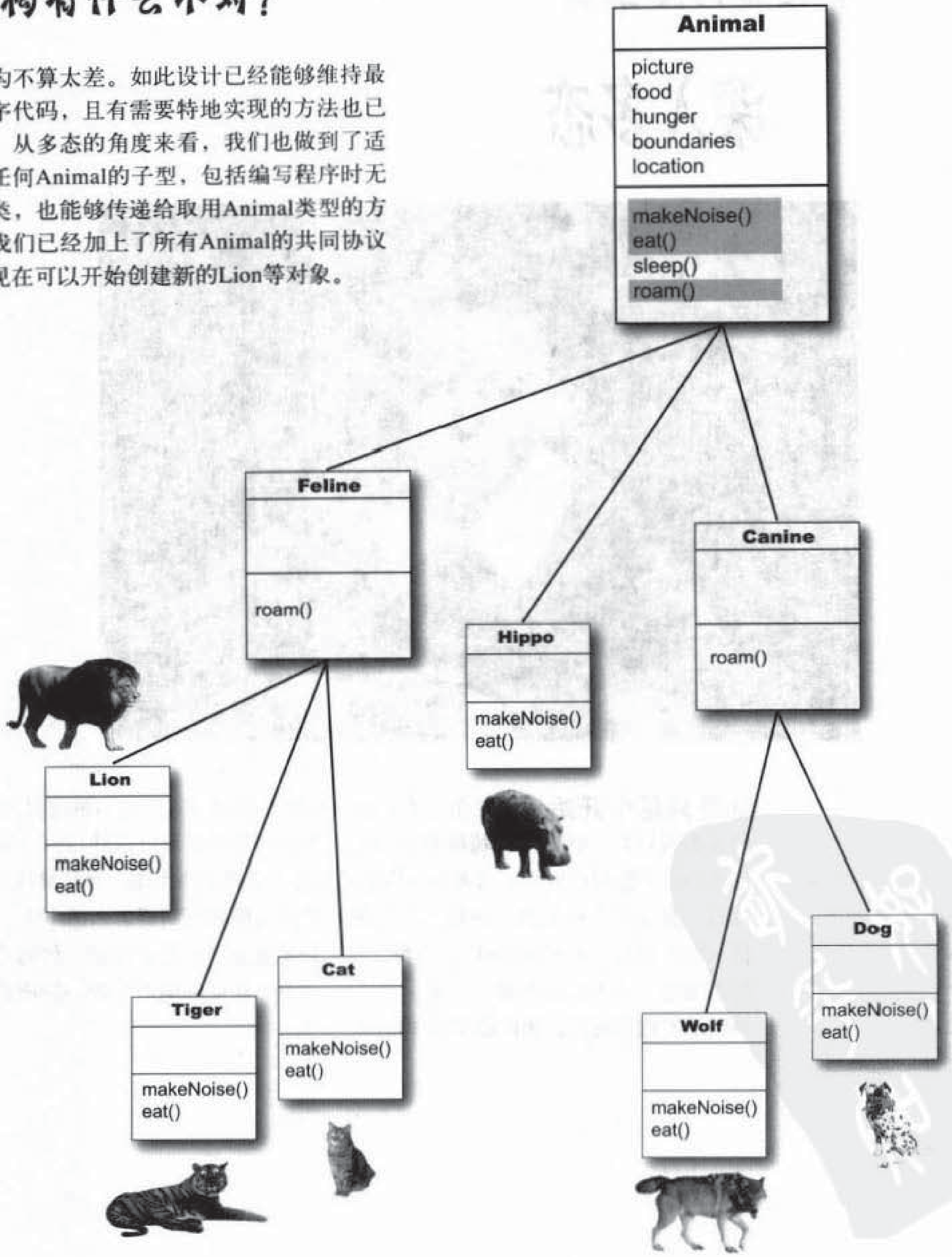
深入多态



继承只是个开始。 要使用多态，我们还需要接口（这个接口的意义不是GUI的I所代表的接口）。我们需要超越简单的继承并前进到只有通过设计与编写接口规格才能达成的适应性与扩展性。很多Java功能若没有接口就无法工作，因此就算你不会去设计接口，也还是会使用到。最后你会怀疑如果没有接口机制要怎么活下去。到底接口是什么呢？它是一种100%纯抽象的类。什么是抽象类？它是无法初始化的类。抽象类有什么用途？马上就会告诉你。前一章让Vet这个方法能够运用Animal的所有子类只是多态的基本招式而已。接口是多态和Java的重点。

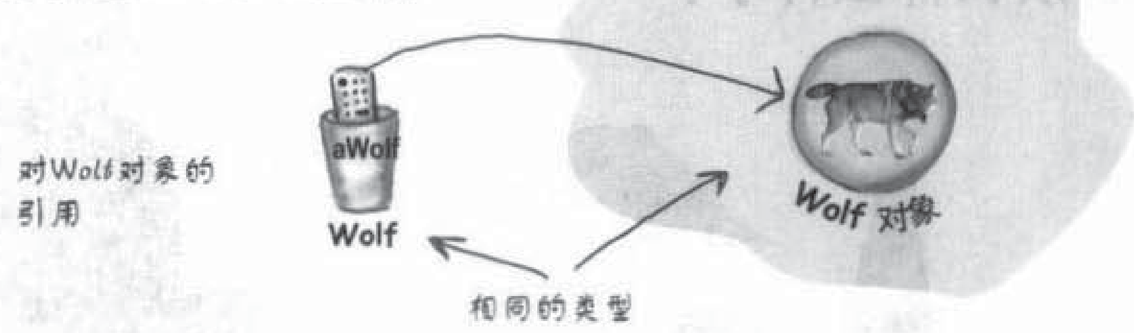
这个结构有什么不对?

这个class结构不算太差。如此设计已经能够维持最少的重复程序代码，且有需要特地实现的方法也已经被覆盖过。从多态的角度来看，我们也做到了适应性，所以任何Animal的子型，包括编写程序时无法想象的种类，也能够传递给取用Animal类型的方法来执行。我们已经加上了所有Animal的共同协议到父类上，现在可以开始创建新的Lion等对象。



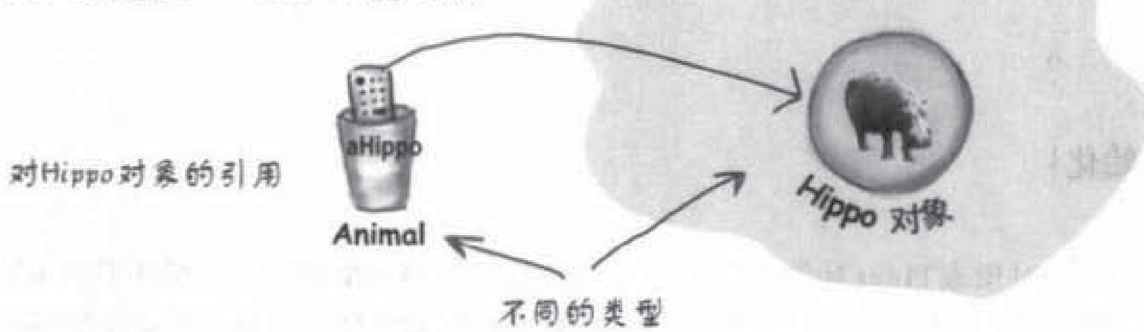
我们可以写出这样的指令：

```
Wolf aWolf = new Wolf();
```



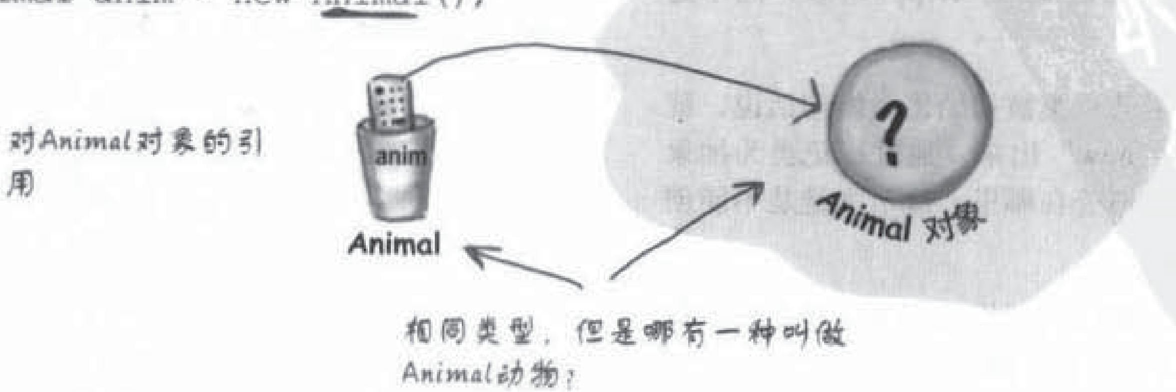
也可以这样：

```
Animal aHippo = new Hippo();
```



但是这样会很奇怪：

```
Animal anim = new Animal();
```



Animal对象应该长什么样子？



怪物？



实例变量的值会是……？

有些类不应该被初始化！

创建出Wolf对象或Hippo对象或Tiger对象是很合理的，但是Animal对象呢？它应该长什么样子？什么颜色、大小、几条腿？

尝试创建出Animal类型的对象就好像出人意料之外，在传送组合的过程中发生了一点问题。

那要如何处理这个问题呢？我们一定要有Animal这个类来继承和产生多态。但是要限制只有它的子类才能够被初始化。我们要的是Lion、Hippo对象，而不是Animal对象。

幸好，有个方法能够防止类被初始化。换句话说，就是让这个类不能被“new”出来。通过标记类为抽象类的，编译器就知道不管在哪里，这个类就是不能创建任何类型的实例。

你还是可以用这种抽象的类型作为引用类型。这也就是当初为何要有抽象类型的目的。

当你设计好继承结构时，你必须要决定哪些类是抽象的，而哪些是具体的。具体的类是实际可以被初始化为对象的。

设计抽象的类很简单 —— 在类的声明前面加上抽象类关键词abstract就好：

```
abstract class Canine extends Animal {
   public void roam() { }
}
```

编译器不会让你初始化抽象类

抽象类代表没有人能够创建出该类的实例。你还是可以使用抽象类来声明为引用类型给多态使用，却不用担心哪个创建该类型的对象，编译器会确保这件事。

```
abstract public class Canine extends Animal
{
   public void roam() { }
}
```

```
public class MakeCanine {
   public void go() {
      Canine c;
      c = new Dog();
      c = new Canine();
      c.roam();
   }
}
```

这是可以的，因为你可以赋值子类对象给父类的引用，即使父类是抽象的

这个类已经被标记为abstract，所以过不了编译器这一关

```
File Edit Window Help BeamMeUp
% javac MakeCanine.java
MakeCanine.java:5: Canine is abstract;
cannot be instantiated
      c = new Canine();
          ^
1 error
```

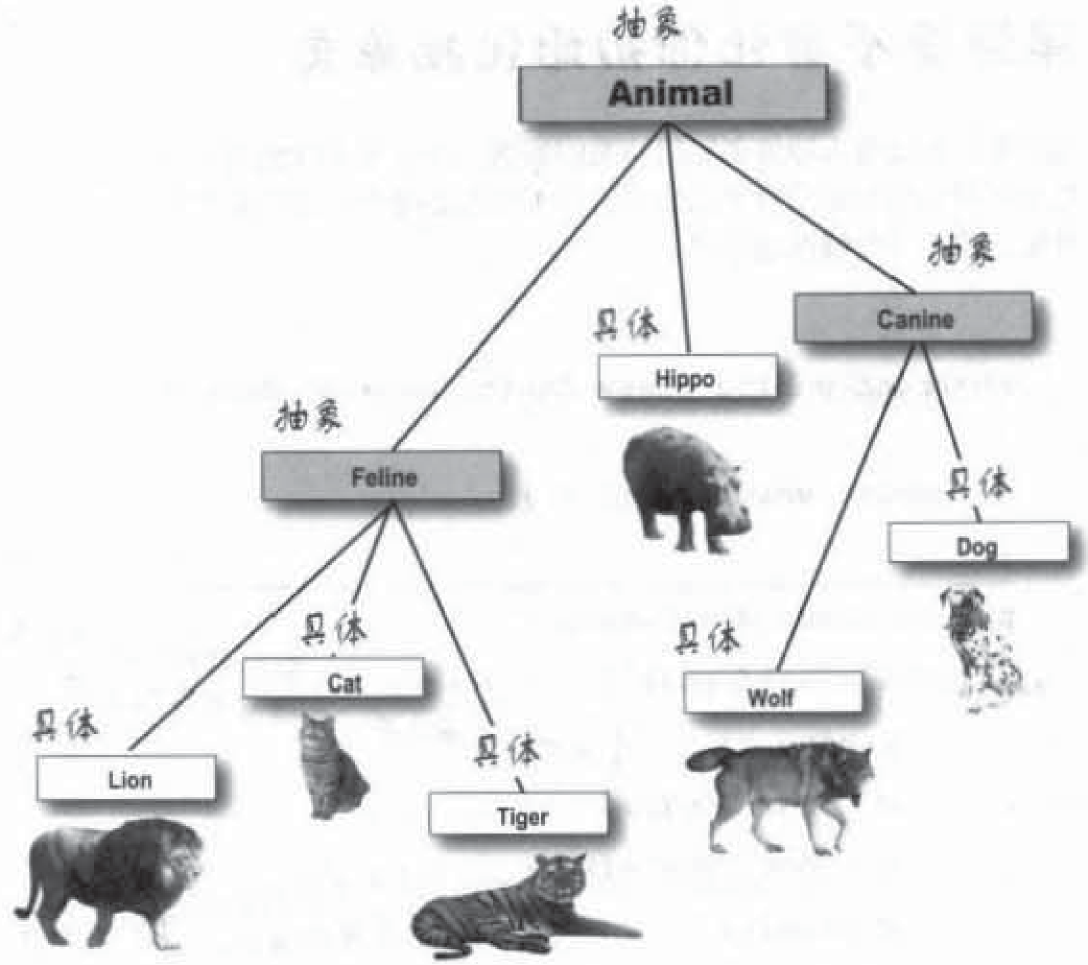
抽象类除了被继承过之外，是没有用途、没有值、没有目的*。

*除了抽象的class可以有static的成员之外，见第10章。

抽象与具体

不是抽象的类就被称为具体类。在Animal的继承树下，如果我们创建出Animal、Canine与Feline的抽象，则Hippo、Wolf等就是具体的类。

查阅Java API你就会发现其中有很多的抽象类，特别是GUI的函数库中更多。GUI的组件类是按钮、滚动条等与GUI有关类的父类。你只会对组件下的具体子类作初始化动作。



BRAIN POWER



抽象或具体?

你怎么知道某个类应该是抽象的？饮料或许是抽象的。那大雕参茸或威士忌是否也应该是抽象的？在继承层次中从哪个点开始才算是具体的？

你会把“提神饮料”设计成具体，还是说它也是个抽象？看起来“保力达 B”才会是具体的。你认为呢？

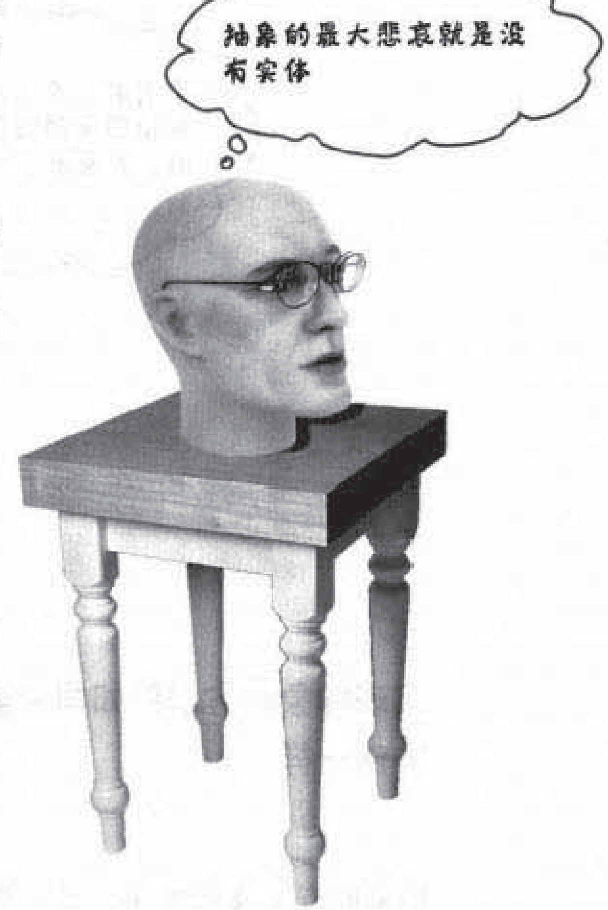
观察上面的Animal继承层次，这些抽象或具体的决定是否合适呢？你会修改这个层次吗？

抽象的方法

除了类之外，你也可以将方法标记为abstract的。抽象的类代表此类必须要被extend过，抽象的方法代表此方法一定要被覆盖过。你或许认为抽象类中的某些行为在没有特定的运行时不会有任何的意义。也就是说，没有任何通用的实现是可行的。想象一下通用的eat()方法会有什么结果？

抽象的方法没有实体！

因为你已经知道编写出抽象方法的程序代码没有意义，所以不会含有方法。




```
public abstract void eat();
```

没有方法体！直接以分号结束

如果你声明出一个抽象的方法，就必须将类也标记为抽象的。你不能在非抽象类中拥有抽象方法。

就算只有一个抽象的方法，此类也必须标记为抽象的。

there are no Dumb Questions

问：为什么要有抽象的方法？我认为抽象类的重点就在于可以被子类继承的共同程序代码。

答： 将可继承的方法体（也就是有内容的方法）放在父类中是个好主意。但有时就是没有办法作出给任何子类都有意义的共同程序代码。抽象方法的意义是就算无法实现出方法的内容，但还是可以定义出一组子型共同的协议。

问：这样做的好处是……？

答： 就是多态！记住，你想达成的目标是要使用父型作为方法的参数、返回类型或数组的类型。通过这个机制，你可以加入新的子型到程序中，却又不必重写或修改处理这些类型的程序。想象一下如果不是使用Animal作为Vet的方法参数程序会写成什么样子……你必须为每一种动物写出不同的方法！因此多态的好处就在于所有子型都会有那些抽象的方法。

你必须实现所有抽象的方法



实现抽象的方法就如同覆盖过方法一样

抽象的方法没有内容，它只是为了标记出多态而存在。这表示在继承树结构下的第一个具体类必须要实现出所有的抽象方法。

然而你还是可以通过抽象机制将实现的负担转给下层。例如说将Animal与Canine都标记为abstract，则Canine就无需实现出Animal的抽象方法。但具体的类，例如说Dog，就得实现出Animal和Canine的抽象方法。

记得抽象类可以带有抽象和非抽象的方法，因此Canine也可以实现Animal的抽象方法，让Dog不必实现这个部分。如果Canine没有对Animal的抽象类表示出任何意见，就表示Dog得自己实现出Animal的抽象方法。

当我们谈到“你必须实现所有抽象的方法”时，表示说你必须写出内容。你必须以相同的方法鉴名（名称与参数）和相容的返回类型创建出非抽象的方法。Java很注重你的具体子类有没有实现这些方法。

Sharpen your pencil

抽象类与具体类

让我们来对抽象修辞学产生些具体的用途。在中间这行列出一些类。你的任务是想象有哪些应用程序会把它们设计成具体的，又有哪些应用程序会把它们设计成抽象的。我们已经举出几个例子，例如Tree这个类在植物看护应用程序中应该是抽象的，而在高尔夫球场仿真程序中应该会是具体的。

具体	类	抽象
高尔夫模拟	Tree	植物看护
______	House	建筑设计程序
卫星影像程序	Town	______
______	Football Player	教练程序
______	Chair	______
______	Customer	______
______	Sales Order	______
______	Book	______
______	Store	______
______	Supplier	______
______	Golf Club	______
______	Carburetor	______
______	Oven	______

多态的使用

假设我们不知道有ArrayList这种类而想要自行编写维护list的类以保存Dog对象。在第一轮我们只会写出add()方法。我们使用大小为5的简单Dog数组（Dog[]）来保存新加入的Dog对象。当Dog对象超过5个时，你还是可以调用add()方法，但是什么事情也不会发生。如果没有越界，add()会把Dog装到可用的数组位置中，然后递增可用索引（nextIndex）。

自己创建的Dog专用list

（这或许是有史以来自行尝试编写ArrayList类型类中最差劲的一个程序）




```
public class MyDogList {
    private Dog [] dogs = new Dog[5];
    private int nextIndex = 0;

    public void add(Dog d) {
        if (nextIndex < dogs.length) {
            dogs[nextIndex] = d;
            System.out.println("Dog added at " + nextIndex);
            nextIndex++;
        }
    }
}
```

实际上使用的是数组

增加新的Dog就加1

如果没有超出上限就把Dog加进去并列出信息

递增计数

糟了，也要写出Cat用的

我们有几个选项：

（1）另外创建一个单独的MyCatList类来处理Cat对象，这不好。

（2）创建一个单独的DogAndCatList类，用 addCat(Cat c) 与addDog(Dog d) 来同时处理两个不同的数组实例，这也不好。

（3）编写一个不同的AnimalList类让它处理Animal所有的子类。这应该是最好的办法，所以我们就这么处理，以更通用的Animal来取代个别的子类。程序变更得关键部分有特别标出来（逻辑还是一样，只是把Dog换成Animal）。

自己创建的Animal通用list

version 2

MyAnimalList
Animal[] animals int nextIndex
add(Animal a)

```
public class MyAnimalList {

    private Animal[] animals = new Animal[5];
    private int nextIndex = 0;

    public void add(Animal a) {
       if (nextIndex < animals.length) {
          animals[nextIndex] = a;
          System.out.println("Animal added at " + nextIndex);
          nextIndex++;

       }
    }
}
```

别紧张，这不是在创建Animal对象，只是个保存Animal的数组对象

```
public class AnimalTestDrive{
   public static void main (String[] args) {
      MyAnimalList list = new MyAnimalList();
      Dog a = new Dog();
      Cat c = new Cat();
      list.add(a);
      list.add(c);
   }
}
```

File Edit Window Help Harm

```
% java AnimalTestDrive
Animal added at 0
Animal added at 1
```

非Animal呢？何不写个万用类？

你知道这要怎么做。我们可以修改数组的类型，并且调整add()方法的参数，以处理Animal之上的类。那便是更通用、更抽象的一种类。但是真的有这种类吗？我们设计的Animal并没有父类不是吗？

事实上是有的。

还记得ArrayList的方法吗？它们是通过对象这个类型来操纵所有类型的对象。

在Java中的所有类都是从Object这个类继承出来的。

Object这个类是所有类的源头；它是所有类的父类。

如果Java中没有共同的父类，那将无法让Java的开发人员创建出可以处理自定义类型的类，也就是说无法写出像ArrayList这样可以处理各种类的类。

就算你不知道，但实际上所有的类都是从对象给继承出来的。你可以把自己写的类想象成是这样声明的：

```
public class Dog extends Object { }
```

但是Dog本来是从Canine给extends出来的啊！没关系，编译器会知道改成让Canine去继承对象。事实上是Animal去继承对象。

没有直接继承过其他类的类会是隐含的继承对象。

所以就算Dog或Canine没有直接extend对象，还是会通过Animal来继承对象。

version 3 （这只是部分的ArrayList中的方法）

ArrayList

boolean remove(Object elem)
根据索引参数移动对象，如果list中没有元素返回true

boolean contains(Object elem)
如果和对象的参数相匹配的的话返回true

boolean isEmpty()
如果list中没有元素返回true

int indexOf(Object elem)
返回对象参数的索引值或-1

Object get(int index)
返回元素在list中的位置

boolean add(Object elem)
向list中增加元素

// more

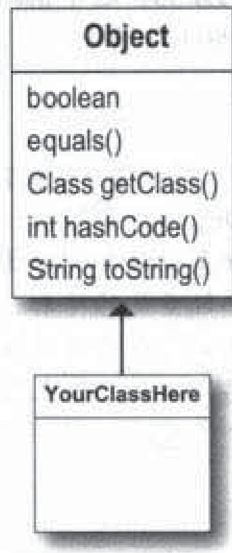
许多ArrayList的方法都用到Object这个终极类型。因为每个类都是对象的子类，所以ArrayList可以处理任何类！

注意：Java 5.0的get()和add()方法与这里所显示的有所不同，但无损于此处要表达的概念。

终极对象有什么？

如果你是Java，那你会想要让每个对象都带有什么行为？嗯……来个可以判断某对象是否与其他对象相等的方法如何？再加上一个可以说明它是什么类的方法怎样？或许还会需要一个产生对象哈希代码的方法？你可以运用哈希表上的对象（我们会在第17章和附录B中讨论哈希表）。

你知道怎样吗？对象的确有上面所说的方法。那还不是全部的方法，但目前我们只关心这几个。



只列出类对象的一部分方法

所有的类都有继承这个类

1 equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();

if (a.equals(c)) {
    System.out.println("true");
} else {
    System.out.println("false");
}
```

```
File Edit Window Help Stop
% java TestObject
false
```

这会让你知道是否两个对象可认为是“相等”的（见附录B）

3 hashCode()

```
Cat c = new Cat();
System.out.println(c.hashCode());
```

```
File Edit Window Help Drop
% java TestObject
8202111
```

列出此对象的哈希代码，你可以把它想成是一个唯一的ID

4 toString()

```
Cat c = new Cat();
System.out.println(c.toString());
```

```
File Edit Window Help LapseIntoComa
% java TestObject
Cat@7d277f
```

列出类的名称和一个我们不关心的数字

2 getClass()

```
Cat c = new Cat();
System.out.println(c.getClass());
```

```
File Edit Window Help Faint
% java TestObject
class Cat
```

告诉你此对象是从哪里被初始化的

there are no Dumb Questions

问： Object这个类是抽象的吗？

答： 不是。至少不是正式的Java抽象类。因为它可以被所有类继承下来的方法都实现程序代码，所以没有必须被覆盖过的方法。

问： 那是否可以覆盖过Object的方法？

答： 部分可以。但是有些被标记为final，这代表你不能覆盖掉它们。强烈建议你用自己写的类去覆盖掉hastCode()、equals()以及toString()。

问： 如果ArrayList方法是通用的，那ArrayList <DotCom>是什么意思？

答： 限制它的类型。在Java 5.0之前无法限制它的类型。如果你写成ArrayList<Dog>，则此ArrayList受限只能保存Dog的对象。这种新型的语法会在后面的章节有更多的说明。

问： Object类是具体的。怎么会允许有人去创建Object的对象呢？这不就跟Animal对象一样不合理吗？

答： 好问题！为何要允许创建出Object的实例呢？因为有时候你就是会需要一个通用的对象，一个轻量化的对象。它最常见的用途是用在线程的同步化上面（见第15章）。你先当作不会用到这个对象。

问： 所以Object的主要目的是提供多态的参数与返回类型吗？

答： 这个Object类有两个主要的目的：作为多态让方法可以应付多种类型的机制，以及提供Java在执行期对任何对象都有需要的方法的实现程序代码（让所有的类都会继承到）。有一部分的方法是与线程有关，这会在后面的章节说明。

问： 即然多态类型这么有用，为什么不把所有的参数和返回类型都设定成Object类型哪？

答： 啊……想想看这会发生什么后果。考虑一下何谓“类型安全检查”。它是Java保护程序代码的一项重要机制。在此机制下，你不会意外地要求对象执行错误类型的动作。例如说防止你对Cefiro要求Ferrai的加速动作。

但事实上，你也不用担心会发生这件事，因为当某个对象是以Object类型来引用时，Java会把它当作Object类型的实例。这代表你只能调用由Object类中所声明的方法。若你像下面这样做：

```
Object o = new Ferrai( );
o.goFast( );//非法
```

则第二行会无法通过编译。

因为Java是类型检查很强的程序语言，编译器会检查你调用的是否是该对象确实可以响应的方法。换句话说，你只能从确实有该方法的类去调用。同样，这也会在后面的章节说明。

使用Object类型的多态引用是会付出代价的……

在你开始以Object类型使用所有超适用性参数和返回类型之前，你应该要考虑到使用Object类型作为引用的一些问题。注意此处的讨论不涉及制作出Object类型的实例，这是在说以Object类型作为引用的其他类型。

当你把对象装进ArrayList<Dog>时，它会被当作Dog来输入与输出：

```
ArrayList<Dog> myDogArrayList = new ArrayList<Dog>();   ← 保存Dog的ArrayList
Dog aDog = new Dog();   ← 新建一个Dog
myDogArrayList.add(aDog);   ← 装到ArrayList中
Dog d = myDogArrayList.get(0);   ← 将Dog赋值给新的Dog引用变量
```

但若你把它声明成ArrayList<Object>时会怎样？如果你打算创建出一个可以保存任何一种 对象的ArrayList 时，你会如此的声明：

```
ArrayList<Object> myDogArrayList = new ArrayList<Object>();   ← 保存对象的ArrayList
Dog aDog = new Dog();   ← 新建一个Dog ┐ 相同的步骤
myDogArrayList.add(aDog);   ← 装到ArrayList中 ┘
```

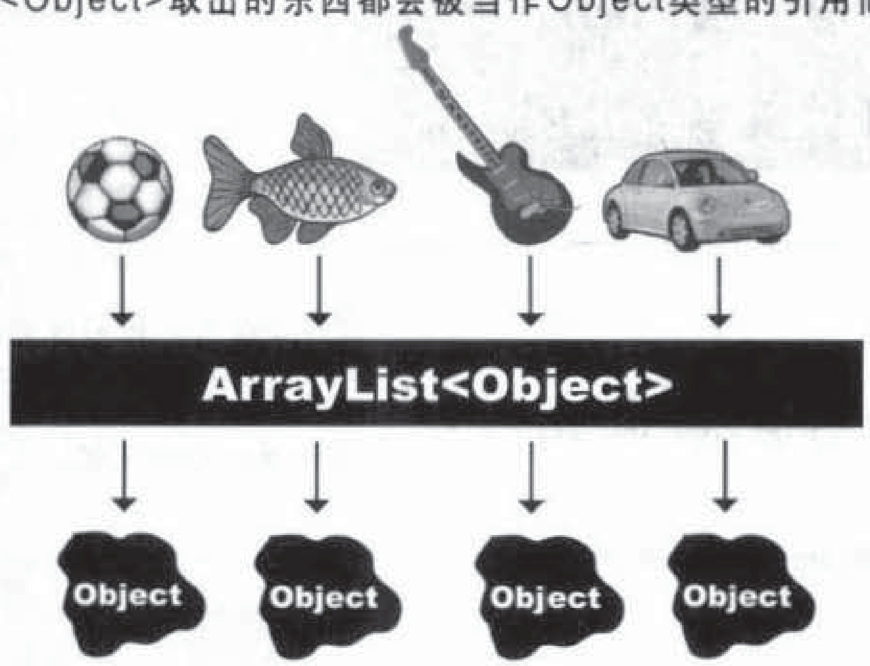
如果是这样，当你尝试要把Dog对象取出并赋值给Dog的引用时会发生什么事？

```
Dog d = myDogArrayList.get(0);
```

无法通过编译！对ArrayList<Object>调用get()方法会返回Object类型，编译器无法确认它是Dog！

任何从ArrayList<Object>取出的东西都会被当作Object类型的引用而不管它原来是什么。

放进去的对象原来是足球、鱼、吉他和汽车



出来后都变成Object

从ArrayList<Object>取出的Object都会被当作是Object这个类的实例。编译器无法将此对象识别为Object以外的事物。

当Dog不再是Dog时



问题在于把所有东西都以多态来当作是Object会让对象看起来失去了真正的本质（但不是永久性的）。Dog似乎失去了犬性。让我们来看一下当我们传入一个Dog给会返回同一个Dog对象的类型引用的方法时会有什么反应。

BAD

```
public void go() {
    Dog aDog = new Dog();
    Dog sameDog = getObject(aDog);
}

public Object getObject(Object o) {
    return o;
}
```

无法过关！虽然这个方法会返回同一个Dog，但编译器认为这只能赋值给Object类型的变量

返回同一个引用，但是类型已经转换成Object了

```
File Edit Window Help Remember
DogPolyTest.java:10: incompatible types
found   : java.lang.Object
required: Dog
      Dog sameDog = getObjects(aDog);
1 error                        ^
```

编译器无法得知方法返回的其实是Dog，因此不会同意这项赋值

GOOD

```
public void go() {
    Dog aDog = new Dog();
    Object sameDog = getObject(aDog);
}

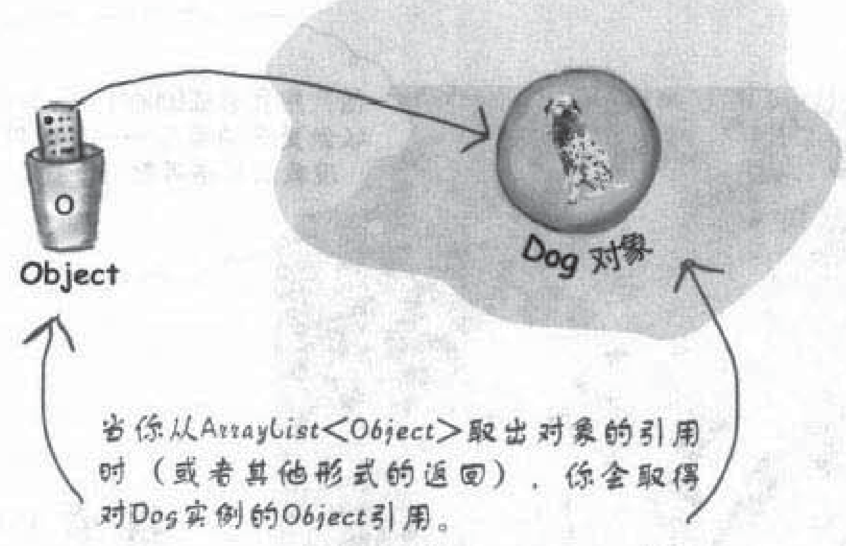
public Object getObject(Object o) {
    return o;

}
```

这样会过关，因为你可以赋值任何东西给Object类型的引用，并且每个东西都能对Object通过IS-A测试。

Object不会吠

我们已经知道当一个对象被声明为Object类型的对象所引用时，它无法再赋值给原来类型的变量。我们也知道这会发生在返回类型被声明为Object类型的时候，例如前面所提过的ArrayList<Object>。但这意味着什么呢？使用Object引用变量来引用Dog对象会是个问题吗？让我们试着对被编译器认为是Object的Dog调用Dog才有的方法看看：




```
Object o = al.get(index);

int i = o.hashCode();

o.bark();
```

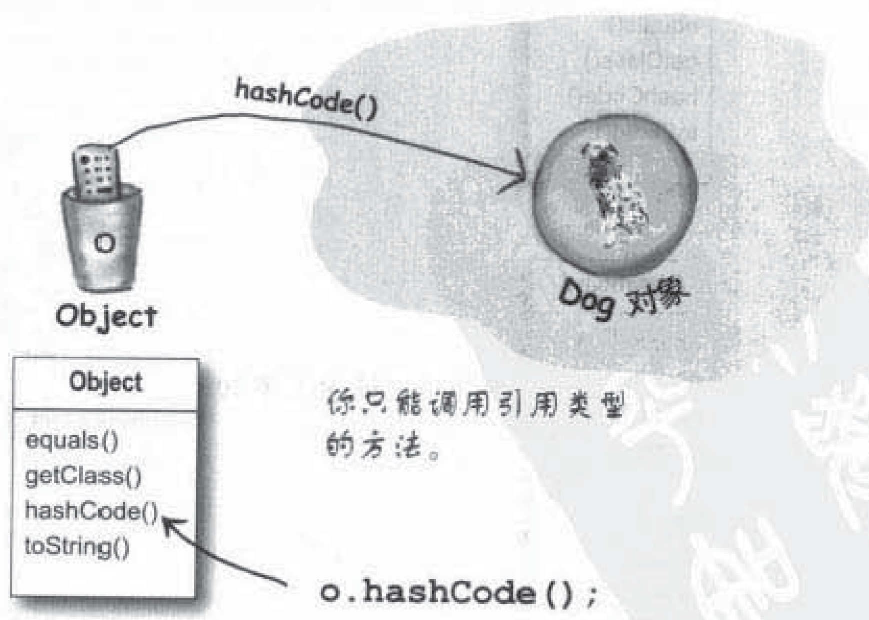
没问题。Object本来就有hashCode()这个方法

有问题！

不能这么做！Object根本就不知道什么是bark()。就算天知地知你知我知但编译器就是不知……

编译器是根据引用类型来判断有哪些method可以调用，而不是根据Object确实的类型。

就算你知道对象有这个功能，编译器还是会把它当作一般的Object来看待。编译器只管引用的类型，而不是对象的类型。

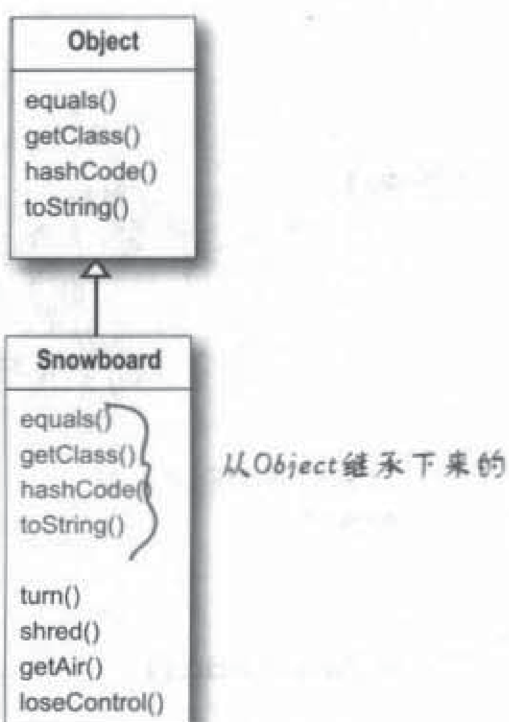


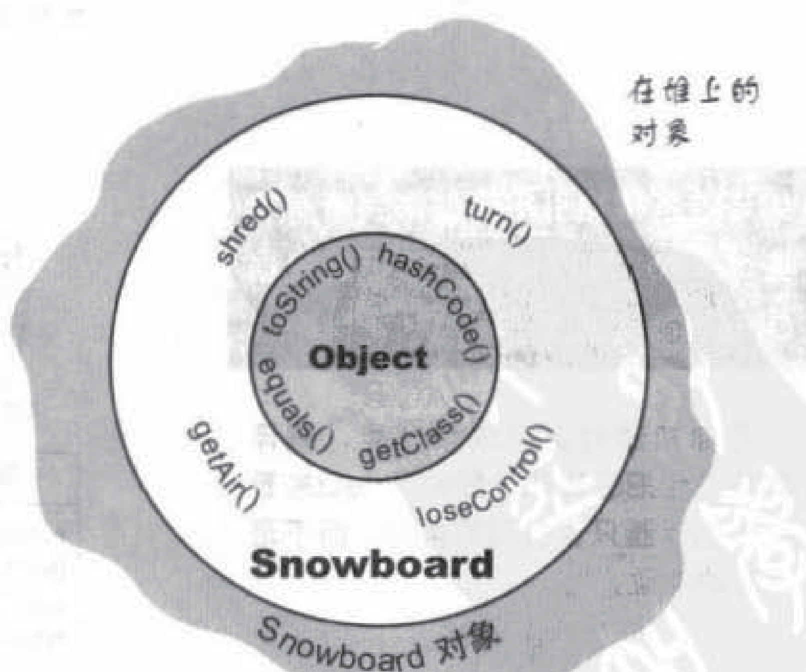
“o”被声明为Object的引用，所以只能通过o调用Object类中的方法。



探索内部Object

对象会带有从父类继承下来的所有东西。这代表每个对象，不论实际类型，也会是个Object的实例。所以Java中的每个对象除了真正的类型外，也可以当作是Object来处理。当你执行new Snowboard()命令时，除了在堆上会有一个Snowboard对象外，此对象也包含了一个Object在里面。





这在堆上只有一个对象，但它实际上带着Snowboard与Object的内容

“多态”意味着“很多形式”

你可以把Snowboard当作Snowboard或者Object

如果引用是个遥控器，则当你在继承树往下走时，会发现遥控器的按钮越来越多。Object的遥控器只有几个按钮而已，但Snowboard的遥控器就会包含有来自Object和自己定义的按钮。越接近具体的类会有越多的按钮。

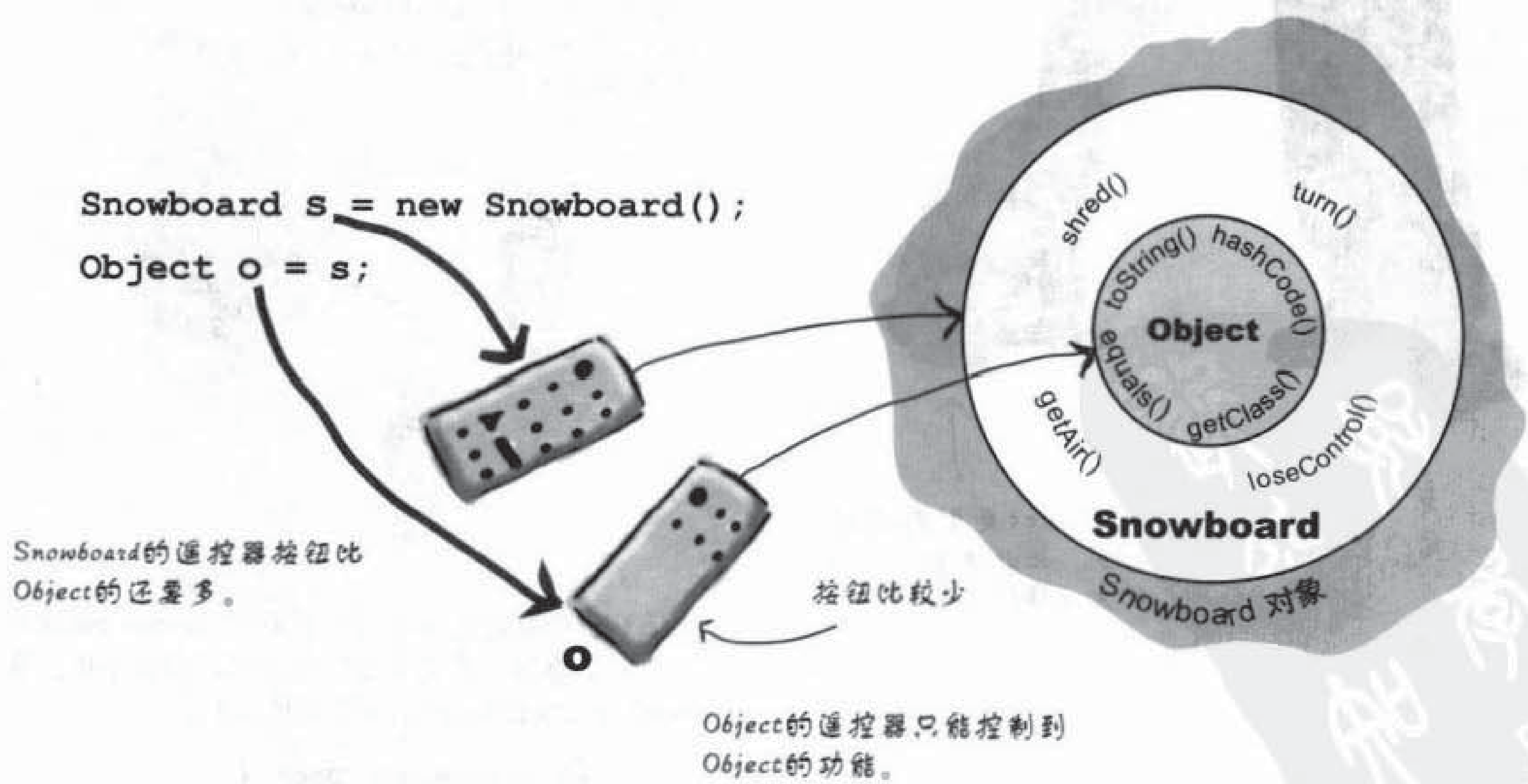
当然这也不是绝对的，子类也有可能不会加入任何新的方法，而只是覆盖过一些方法罢了。重点在于如果对象的类型是Snowboard，而引用它的却是Object，则它不能调用Snowboard的方法。

当你把对象装进ArrayList<Object>时，不管它原来是什么，你只能把它当作是Object。

从ArrayList<Object>取出引用时，引用的类型只会是Object。

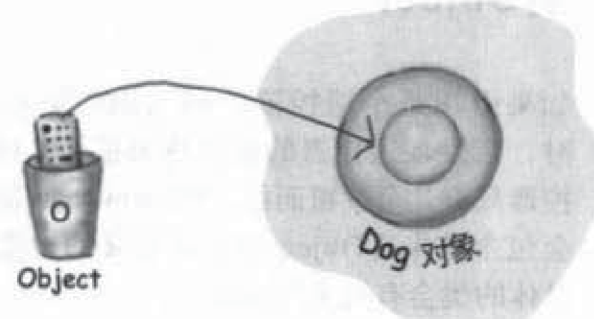
这代表你只会取得Object的遥控器。

```
Snowboard s = new Snowboard();
Object o = s;
```



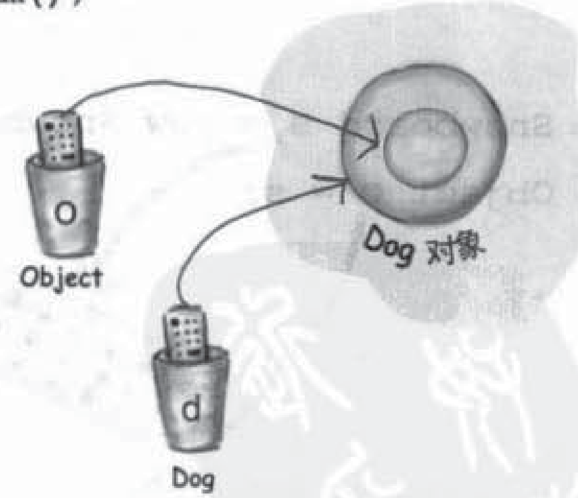


转换回原来的类型



它还是个Dog对象，但如果你想要调用Dog特有的方法，就必须要将类型声明为Dog。如果你真的确定它是个Dog，那么你就可以从Object中拷贝出一个Dog引用，并且赋值给Dog引用变量。

```
Object o = al.get(index);
Dog d = (Dog) o;  ← 将类型转换成Dog
d.roam();
```



如果不能确定它是Dog，你可以使用instanceof这个运算符来检查。若是类型转换错了，你会在执行期遇到ClassCastException异常并且终止。

```
if (o instanceof Dog) {
   Dog d = (Dog) o;
}
```

现在你知道Java是多么注重引用变量的类型。

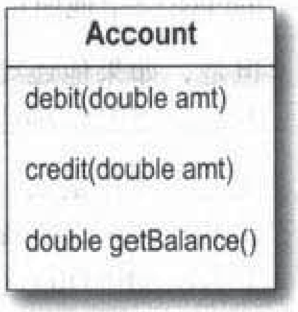
你只能在引用变量的类确实有该方法才能调用它。

把类的公有方法当作是合约的内容，合约是你对其他程序的承诺协议。



在编写类时，大多数情况下你一定会显露出一些方法给类以外的程序使用。要让方法显露就代表你会让方法能够存取得到，通常这会通过标记成公有来完成。

想象这样的情境：你在编写一个小型会计总账系统给“猪标服装社”使用。你发现有个称为Account的类已经写好并且符合你的需求（上一次帮“宏昌水电行”开发程序时写出来的），因此就把它拿过来用。



它有一个debit()和credit()方法可以用来执行会计的借贷项目，还有getBlance()方法可以计算账户。

所以你可以声明一个变量a引用到Account的实例，然后通过圆点运算符调用a.debit()或a.credit()等。

因为类的合约是这么保证的，所以你在这个类的实例上面一定能找到这些方法。

万一想要修改合约呢?

好吧，假如你是个Dog（但是千万别去闻另外一个 Dog 的屁股，这样很不礼貌），你会发现不只有一份合约，你还继承了所有父类传递下来的方法。

类Canine中的所有元素是你合约中的一部分。

类Animal中的所有元素是你合约中的一部分。

类Object中的所有元素是你合约中的一部分。

根据IS-A测试，你就会是Canine、Animal和Object。

如果有人要编写类似的程序，你大可把定义好的class交给他使用。

但是，如果他还要加上亲热或耍宝等宠物特有的功能要怎么办呢?

现在假设你是设计Dog类的程序设计师。没问题吧？你可以直接把beFriendly()和play()这两个方法加进Dog这个类中。这样做不会让其他用到Dog的程序产生问题，因为你没有更改到其他现有的方法。

你觉得这样的做法（把Pet的方法直接加到Dog上）有没有缺点?

BRAIN POWER

如果你是Dog类的程序设计师，且必须修改Dog类以让它能够执行Pet的动作，那你会怎么办？我们知道直接加入Pet的方法是可行的，并且这也不会对其他程序有影响。

但若Cat也要有Pet的功能怎么办？先不管Java的功能，想象一下你要怎样让Animal可以选择性地带有Pet的行为又不会强迫让狮子老虎都表现成宠物?

停下来想想看，动点脑筋会帮助你消耗能量。

有哪些方法可以在PetShop程序中重用现有的类？

在接下来的几页中，我们会对每种可能的方法逐个介绍。先不用管Java实际的功能是怎么做的。一旦知道各种方法的好处与坏处之后，我们就会知道怎么办了。

① 方法一

采用最简单的做法，把宠物方法加进Animal类中。

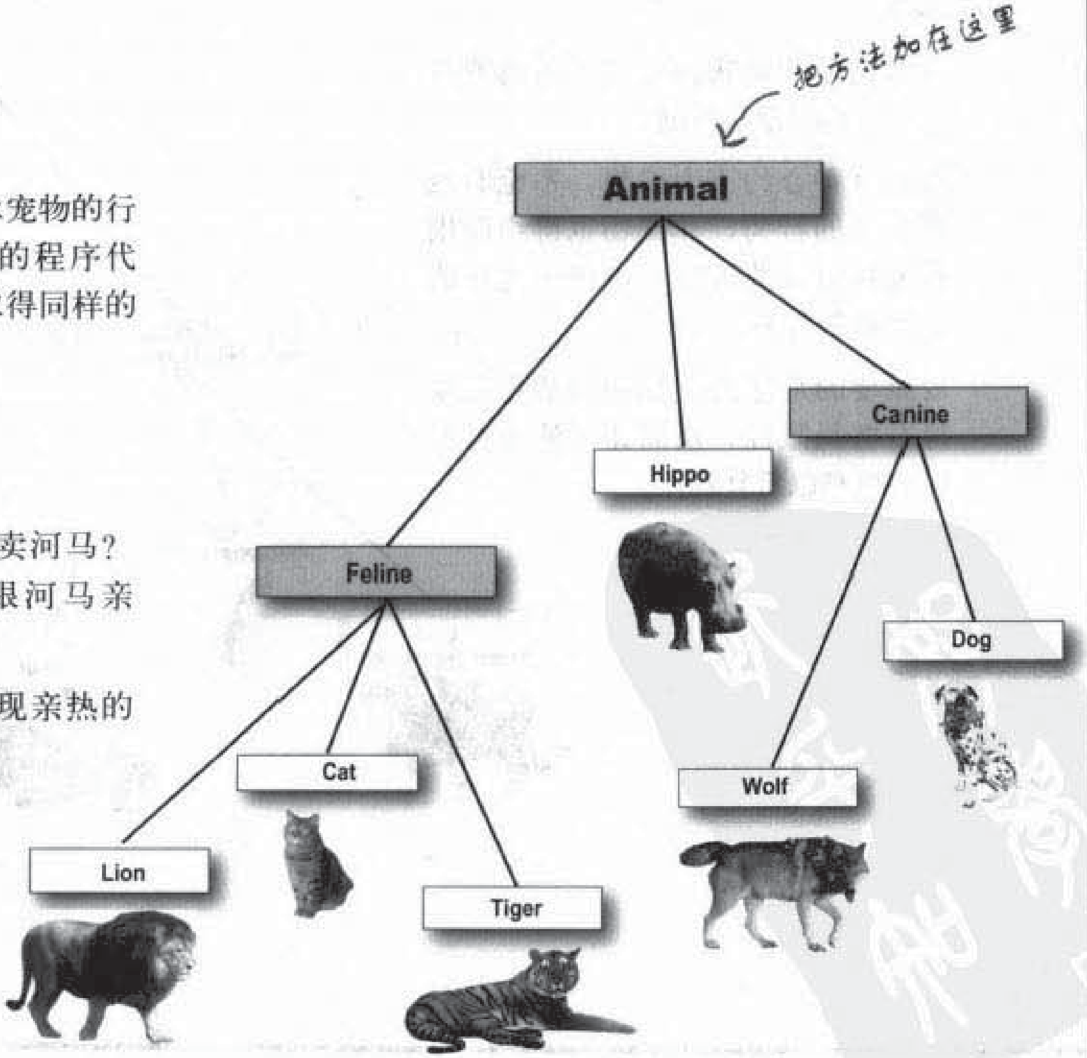
优点：

所有的动物马上就可以继承宠物的行为。不需要改变所有子类的程序代码，而新增加的动物也会取得同样的行为。

缺点：

你什么时候看过宠物店贩卖河马？又是什么时候看到饲主跟河马亲热，带河马去公园散步？

并且我们也知道狗跟猫表现亲热的方式不太一样啊。

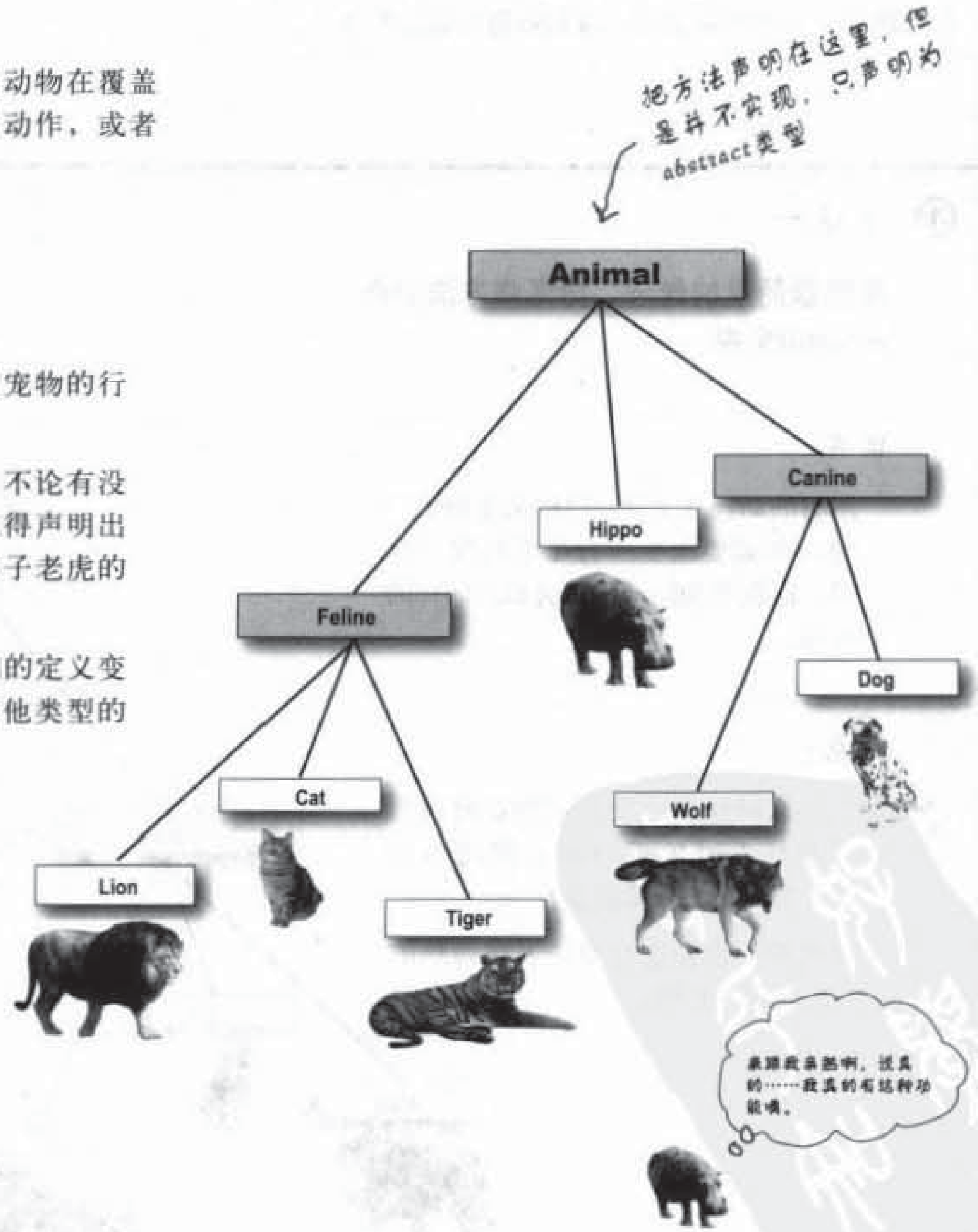


② 方法二

采用方法一，但是把宠物的方法设定成抽象的，强迫每个动物子类覆盖它们。

优点：

这样就可以让非宠物类的动物在覆盖这些方法时，作出合理的动作，或者是什么也不作。



缺点：

所有具体的动物都得实现宠物的行为，这样很浪费时间。

并且这种合约不太理想，不论有没有实质的行为，非宠物也得声明出有宠物行为的外观，对狮子老虎的尊严是个打击。

最重要的是这会让Animal的定义变得有些局限性，反而让其他类型的程序更难以重复利用。

③ 方法三

把方法加到需要的地方。

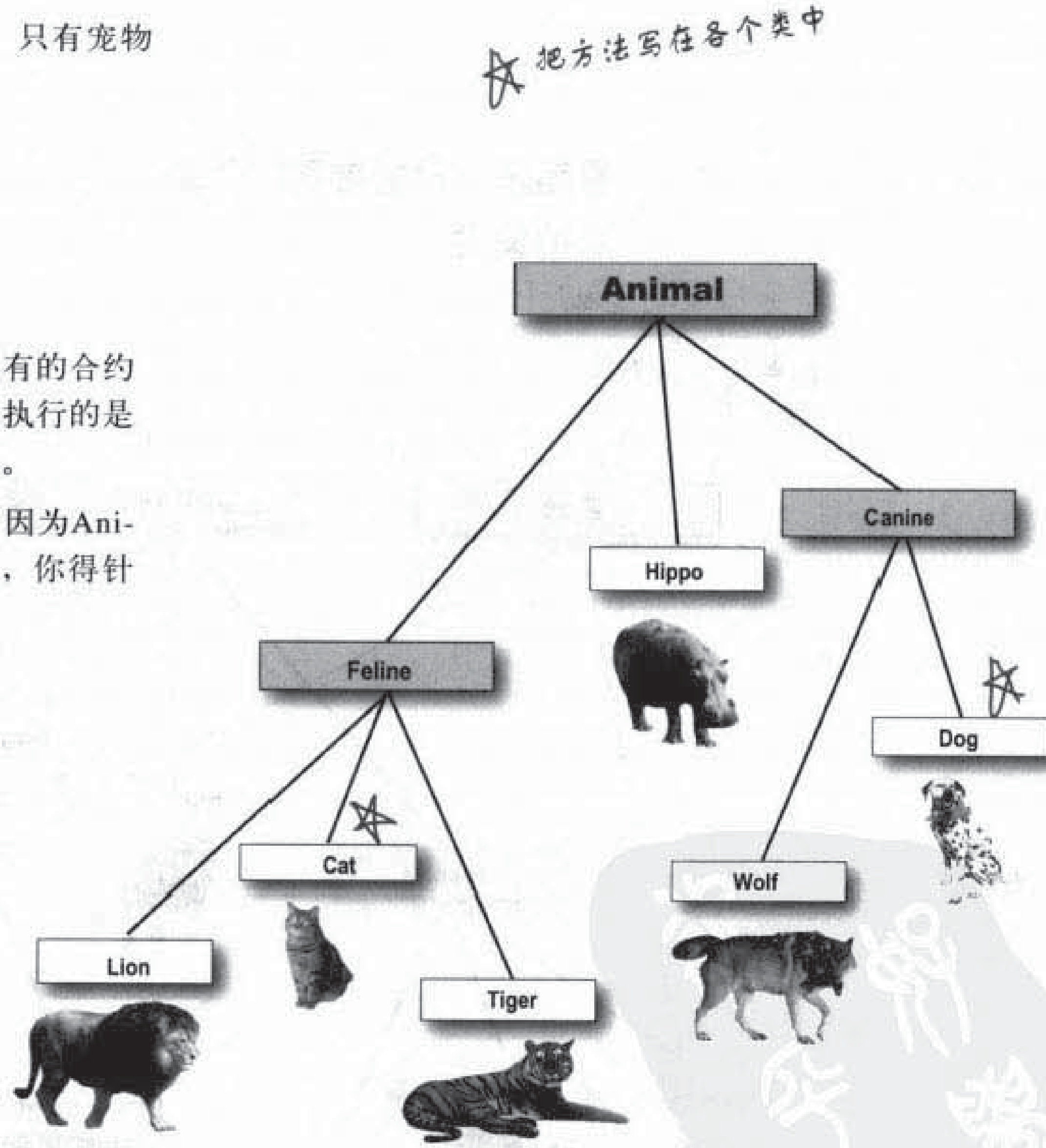
优点：

不必担心如何跟河马亲热。只有宠物才会有宠物的行为。

缺点：

首先，这样就会失去了物该有的合约保证。你无法确定宠物可以执行的是doFriendly()还是beFriendly()。

其次，多态将无法起作用，因为Animal不会有共同的宠物行为，你得针对个别宠物设计程序。



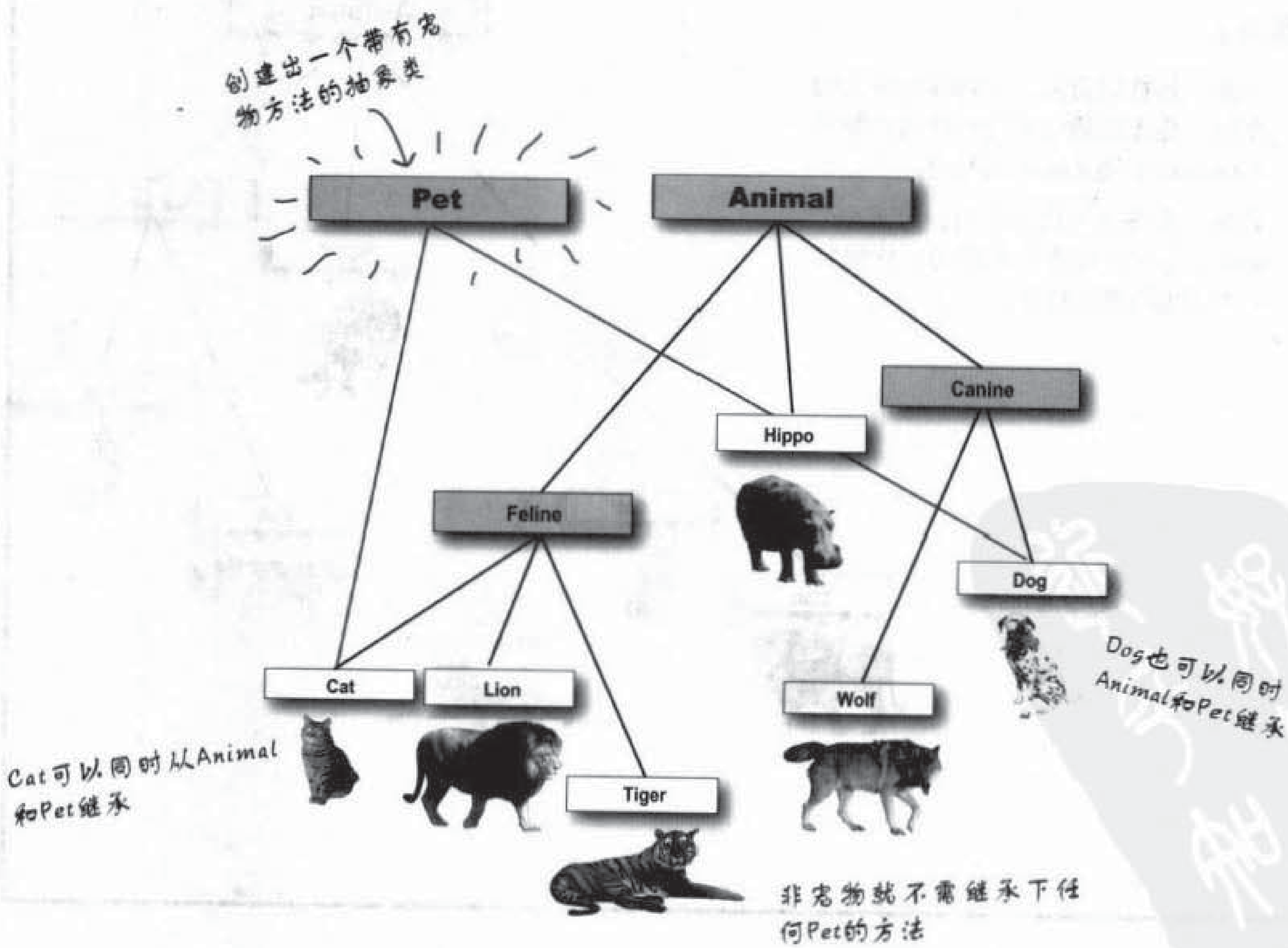
所以我们真正需要的方法是：

(1) 一种可以让宠物行为只应用在宠物身上的方法。

(2) 一种确保所有宠物的类都有相同的方法定义的方法。

(3) 一种是可以运用到多态的方法。

看起来，我们需要两个上层的父类



"两个父类"这个主意有一个问题……

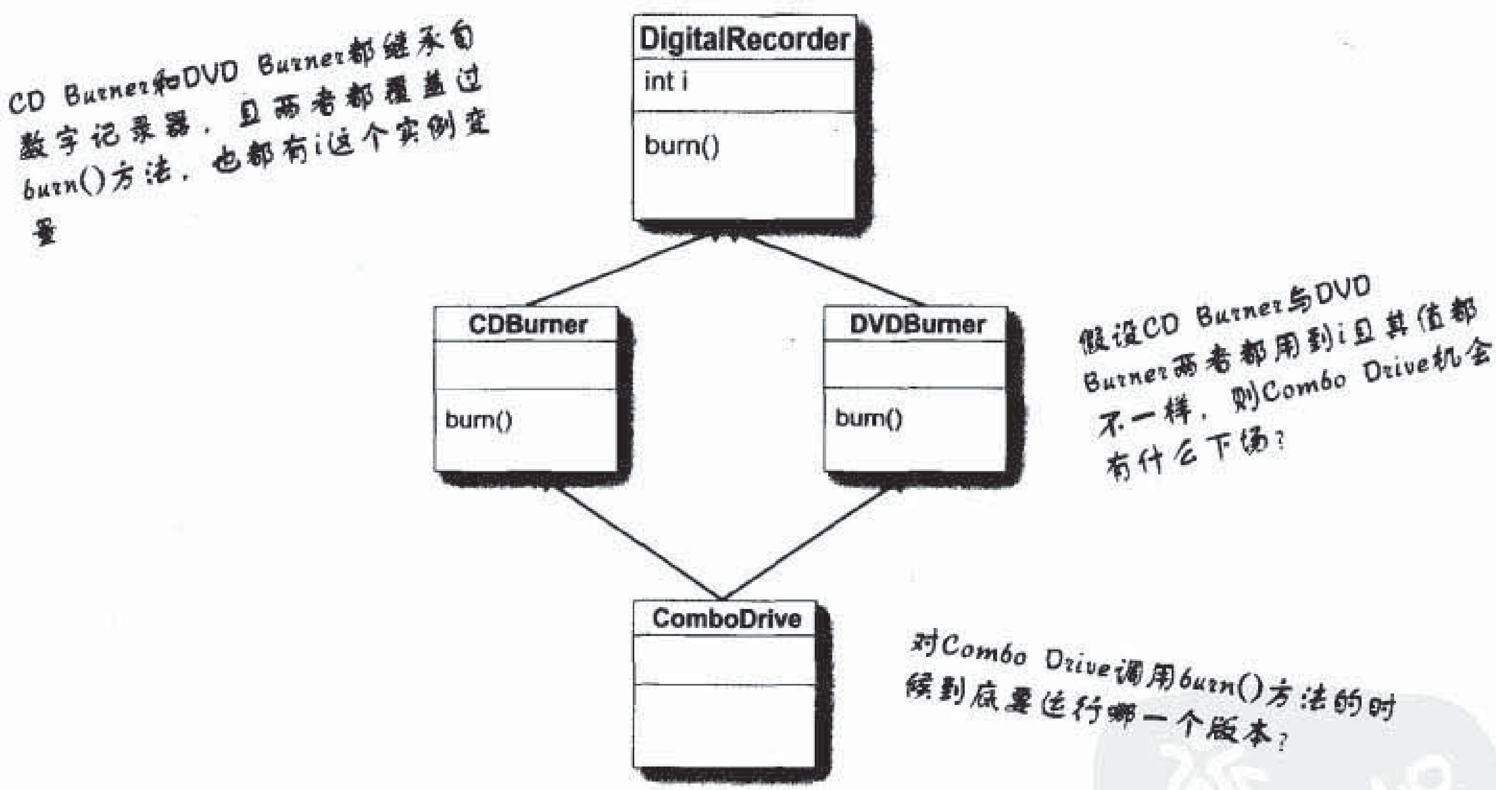
这种"多重继承"可能会很差。

其实Java不支持这种方式，

因为多重继承会有称为"致命方块"的问题。

"致命方块"

（因为这个形状看起来就像扑克牌的方块）



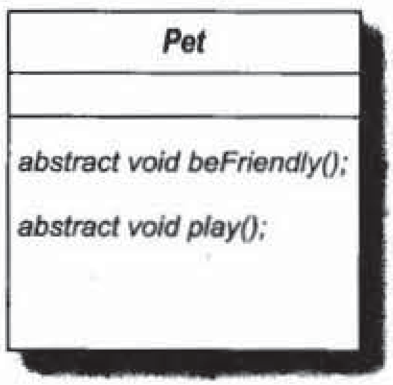
允许致命方块的程序语言会产生某种很糟糕的复杂性问题，因为你必须要有某种规则来处理可能出现的模糊性。额外的规则意味着你必须同时学习这些规则与观察适用这些规则的特殊状况。因此 Java 基于简单化的原则而不允许这种致命方块的出现。好吧，问题还是没有解决……

接口是我们的救星！

Java有个解决方案，使用接口。此处所讨论的不是GUI的接口，也不是“沟通管道”或“存取途径”的接口，我们说的是Java的interface关键词。

此接口可以用来解决多重继承的问题却又不会产生致命方块这种问题。

接口解决致命方块的办法很简单：把全部的方法设为抽象的！如此一来，子类就得要实现此方法，因此Java虚拟机在执行期间就不会搞不清楚要用哪一个继承版本。



Java的接口就好像是100%的纯抽象类。

所有接口的方法都是抽象的，所以任何Pet的类都必须要实现这些方法

接口的定义：

```
public interface Pet {...}
```

使用“interface”来取代“class”

接口的实现：

```
public class Dog extends Canine implements Pet {...}
```

使用implements这个关键词。注意到实现interface时还是必须在某个类的继承之下

设计与实现Pet接口

以interface取代class

接口方法带有public和abstract的意义，这两个修饰符是属于选择性的（我们是为了强调才把它们打出来的，实际上不需要）

```
public interface Pet {
    public abstract void beFriendly();
    public abstract void play();
}
```

接口的方法一定是抽象的，所以必须以分号结束。记住，它们没有内容！

Dog IS-A Animal且
DOG IS-A Pet

implements关键词后面跟着接口的名称

```
public class Dog extends Canine implements Pet {
   public void beFriendly() {...}
   public void play() {..}

   public void roam() {...}
   public void eat() {...}
}
```

必须在这里实现出Pet的方法，这是合约的规定

一般的覆盖方法

there are no Dumb Questions

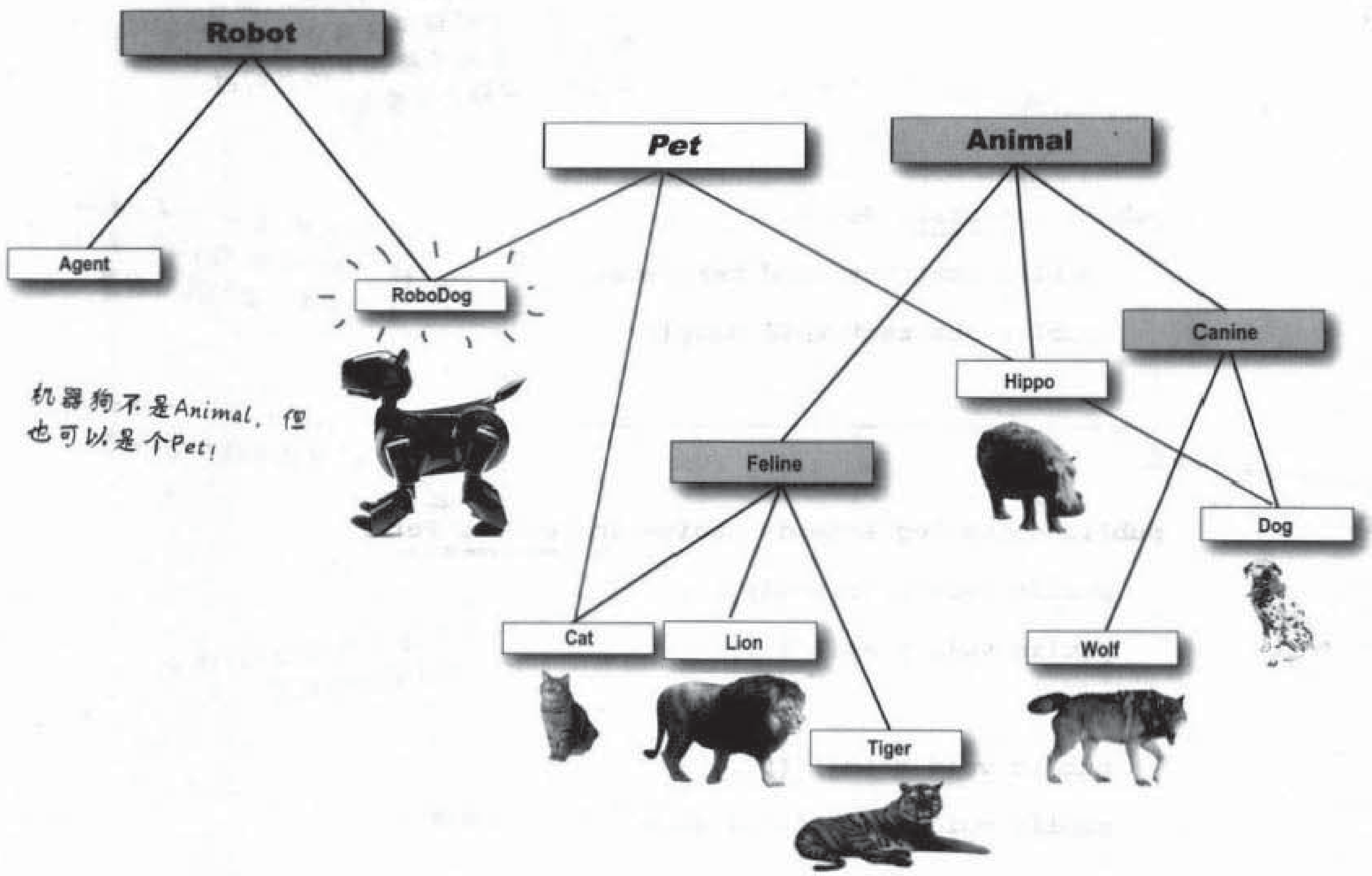
问： **等一下！接口并不是真正的多重继承，因为你无法在它里面实现程序代码，不是吗？如果是这样，那还要接口做什么？**

答： 多态、多态、多态。接口有无比的适用性，若你以接口取代具体的子类或抽象的父类作为参数或返回类型，则你就可以传入任何有实现该接口的东西。这么说吧，使用接口你就可以继承超过一个以上的来源。类可以 extend 过某个父类，并且实现其他的接口。同时其他的类也可以实现同一个接口。因此你就可以为不同的需求组合出不同的继承层次。

事实上，如果使用接口来编写程序，你就是在说："不管你来自哪里，只要你实现这个接口，别人就会知道你一定会履行这个合约"。

大部分良好的设计也不需要在抽象的层次定义出实现细节，我们所需的只是个共同的合约定义。让细节在具体的子类上实现也是很合理的。

不同继承树的类也可以实现相同的接口



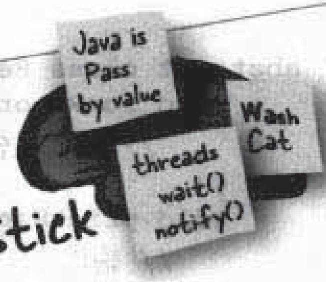
当你把一个类当作多态类型运用时，相同的类型必定来自同一个继承树，而且必须是该多态类型的子类。定义为Canine类型的参数可以接受Wolf与Dog，但无法忍受Cat或Hippo。

但当你用接口来作为多态类型时，对象就可以来自任何地方了。唯一的条件就是该对象必须是来自有实现此接口的类。允许不同继承树的类实现共同的接口对Java API来说是非常重要的。如果你想要将对象的状态保存在文件中，只要去实现Serializable这个接口就行。打算让对象的方法以单独的线程来执行吗？没问题，实现Runnable。有概念了吧。后面的章节会有关于Serializable与Runnable的讨论，现在只要先掌握住这个概念就行。

更棒的是类可以实现多个接口！

通过继承结构，Dog对象IS-A Canine、IS-A Animal、IS-A Object。但Dog IS-A Pet是通过接口实现的机制达成的，并同时也能够实现其他的接口：

```
public class Dog extends Animal implements
Pet, Saveable, paintable {...}
```



Make it Stick

玫瑰是红的，
自我感觉是良好的，
extend只能有一个，implement可以有好多个。
类来自单亲家庭(superclass)，但可以扮演多重角色(implement)。

要如何判断应该是设计类、子类、抽象类或接口呢？

- 如果新的类无法对其他的类通过IS-A测试时，就设计不继承其他类的类。
- 只有在需要某类的特殊化版本时，以覆盖或增加新的方法来继承现有的类。
- 当你需要定义一群子类的模板，又不想让程序员初始化此模板时，设计出抽象的类给它们用。
- 如果想要定义出类可以扮演的角色，使用接口。

调用父类的方法

问： **如果创建出一个具体的子类且必须要覆盖某个方法，但又需要执行父类的方法时要怎么办？也就是说不打算完全地覆盖掉原来的方法，只是要加入额外的动作要怎么做？**

答： 呃……想想看“extends”的字义。设计良好的面向对象要注意到如何编写出必须被覆盖的程序代码。换言之，就是在抽象的类中编写能够共同的实现，让子类加入其余特定的部分。super这个关键词能让你在子类中调用子类的方法。

```
abstract class Report {
    void runReport() {
        // 设置报告
    }
    void printReport() {
        // 输出
    }
}

class BuzzwordsReport extends Report {

    void runReport() {
        super.runReport();
        buzzwordCompliance();
        printReport();

    }
    void buzzwordCompliance() {...}
}
```

父类的方法，带有子类可以运用的部分（指向 `void runReport()`）

调用父版的方法（指向 `super.runReport();`）

如果在子类中指定下面的命令：

super.runReport();

父类的方法就会执行。

super.runReport();

对子类的对象调用会去执行子类覆盖过的方法，但在子类中可以去调用父类的方法



super关键字是用来引用父类对象

要点

- 如果不想让某个类被初始化，就以abstract这个关键词将它标记为抽象的。
- 抽象的类可以带有抽象和非抽象的方法。
- 如果类带有抽象的方法，则此类必定标识为抽象的。
- 抽象的方法没有内容，它的声明是以分号结束。
- 抽象的方法必须在具体的类中运行。
- Java所有的类都是Object（java.lang.Object）直接或间接的子类。
- 方法可以声明Object的参数或返回类型。
- 不管实际上所引用的对象是什么类型，只有在引用变量的类型就是带有某方法的类型时才能调用该方法。
- Object引用变量在没有类型转换的情况下不能赋值给其他的类型，若堆上的对象类型与所要转换的类型不兼容，则此转换会在执行期产生异常。

  类型转换的例子：`Dog d = (Dog) x.getObject(aDog);`
- 从ArrayList<Object>取出的对象只能被Object引用，不然就要用类型转换来改变。
- Java不允许多重继承，因为那样会有致命方块的问题。
- 接口就好像是100%纯天然抽象类。
- 以interface这个关键词取代class来声明接口。
- 实现接口时要使用implements这个关键词。

  例如：`Dog implements Pet`
- class可以实现多个接口。
- 实现某接口的类必须实现它所有的方法，因为这些方法都是public与abstract的。
- 要从子类调用父类的方法可以用super这个关键词来引用。

  例如：`super.RunReport( );`

问： **还是有点怪怪的，你没有解释为何ArrayList<DOG>返回的引用无需转换，却还是在方法中使用Object而不是Dog。使用ArrayList <Dog>时是否有什么怪招？**

答： 说它是怪招一点也不为过。事实上ArrayList根本就不认识 Dog，所以不必作类型转换是个怪招没错。

最简单的回答是：编译器帮你做了类型转换！<Dog>对编译器来说是个禁止将Dog类型以外的对象装进ArrayList的标记。就因为这样，所以编译器也很清楚将从此ArrayList中取出的对象转换为Dog类型是绝对安全的。

但这里面还有很多细节，我们会在讨论Collection的章节加以说明。

练习

这是挑战艺术天分的好机会。下面左方有一组类和接口的声明。你的任务是在右边画出类的图表。第一张图已经帮你画好了。使用虚线来表示实现并以实线来表示继承。

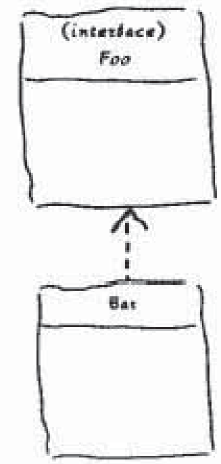
已知：

图呢？

1)

```
public interface Foo { }
public class Bar implements Foo { }
```

1)



2)

```
public interface Vinn { }
public abstract class Vout implements Vinn { }
```

2)

3)

```
public abstract class Muffie implements Whuffie { }
public class Fluffie extends Muffie { }
public interface Whuffie { }
```

3)

4)

```
public class Zoop { }
public class Boop extends Zoop { }
public class Goop extends Boop { }
```

4)

5)

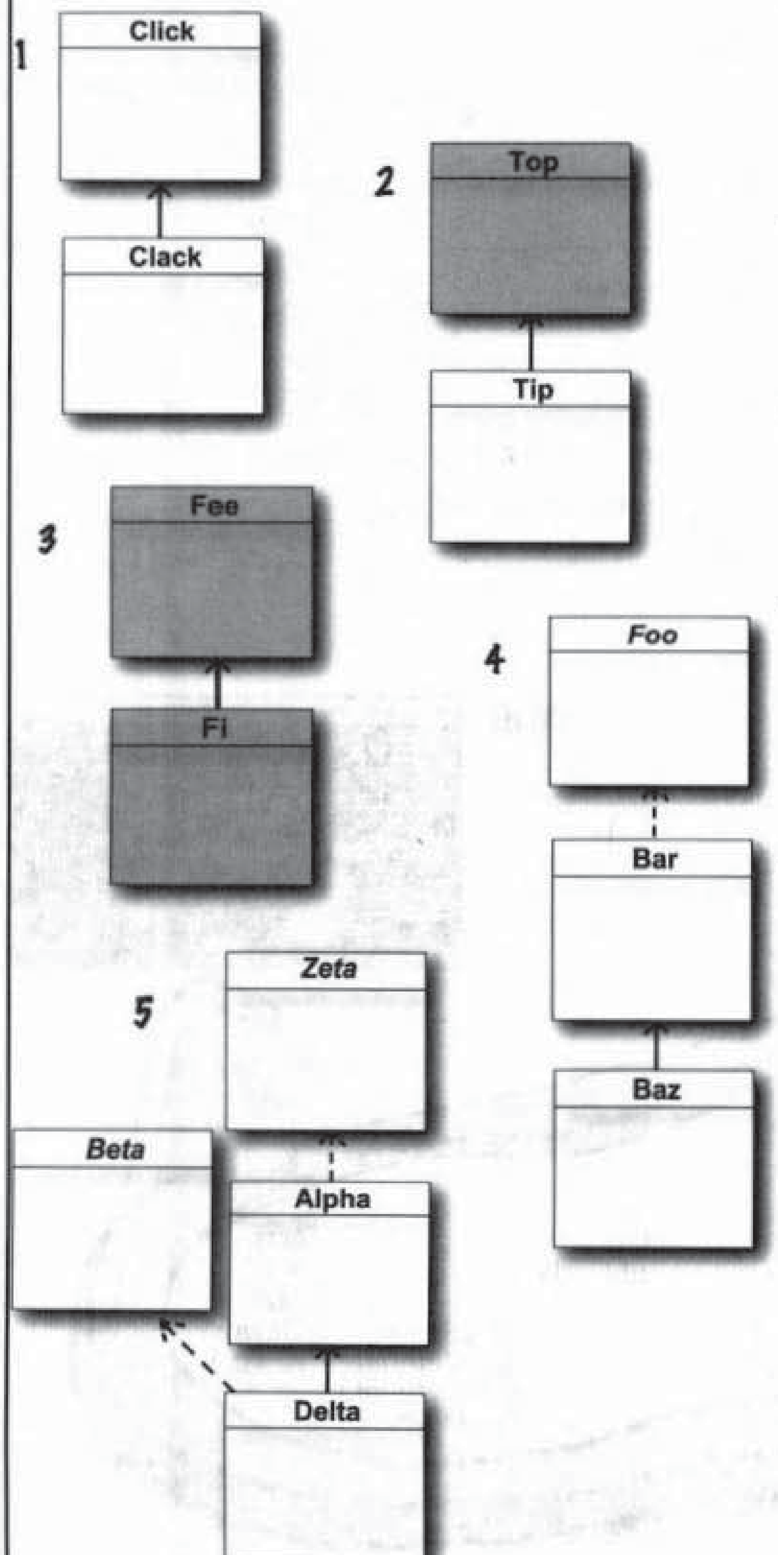
```
public class Gamma extends Delta implements Epsilon { }
public interface Epsilon { }
public interface Beta { }
public class Alpha extends Gamma implements Beta { }
public class Delta { }
```

5)

练习

下面左方有一组class和interface的图表。你的任务是在右边写出有效的Java声明。第一张图已经帮你写好声明。

已知：



Java声明呢？

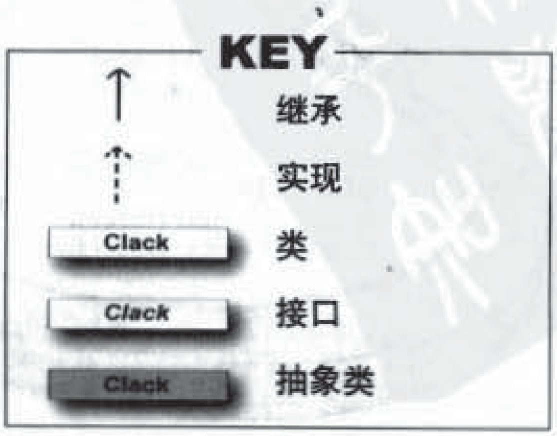
1)
```
public class Click { }
public class Clack extends Click { }
```

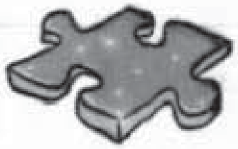
2)

3)

4)

5)





泳池迷宫



你的任务是从游泳池中挑出程序片段并将它们填入右边的空格中。同一个片段可以重复使用，且泳池中有些多余的片段。填完空格的程序必须要能够编译与执行并产生出下面的输出。

```
__________ Nose {
  ____________________
}

abstract class Picasso implements ______{
  ____________________
    return 7;
  }
}

class __________ __________ __________ { }

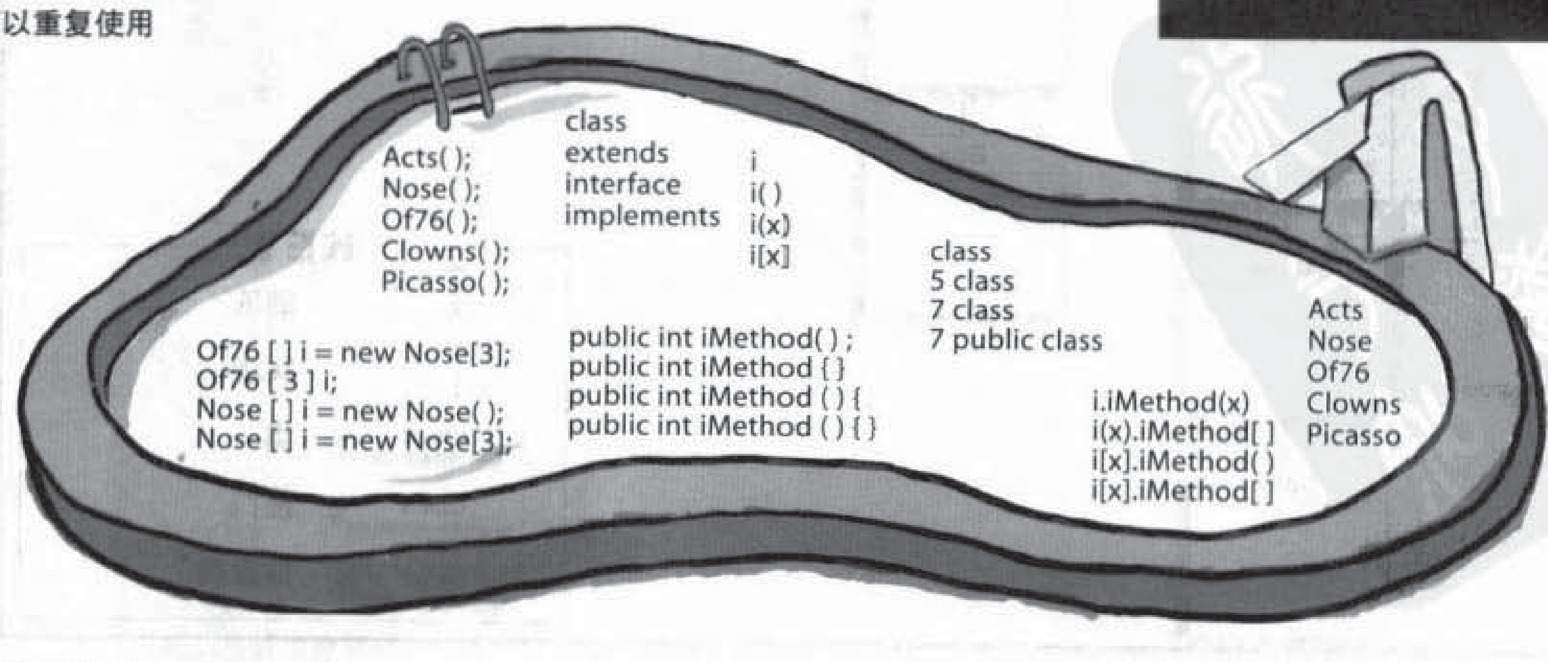
class __________ __________ __________ {
  ____________________
    return 5;
  }
}
```

```
public __________ __________ extends Clowns {
  public static void main(String [] args) {
    ____________________
    i[0] = new __________
    i[1] = new __________
    i[2] = new __________
    for(int x = 0; x < 3; x++) {
      System.out.println(__________
           + " " + _______.getClass( ) );
    }
  }
}
```

输出

```
File Edit Window Help BeAfraid
%java __________
5 class Acts
7 class Clowns
_________Of76
```

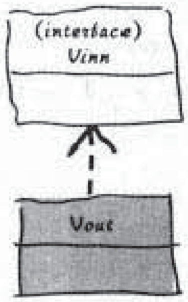
注意：每一个片段可以重复使用



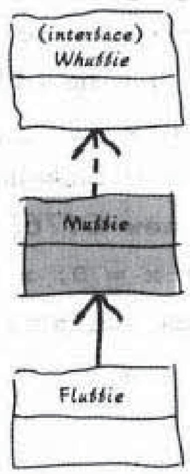
练习解答

艺术天分

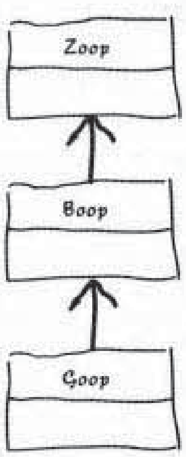
2)



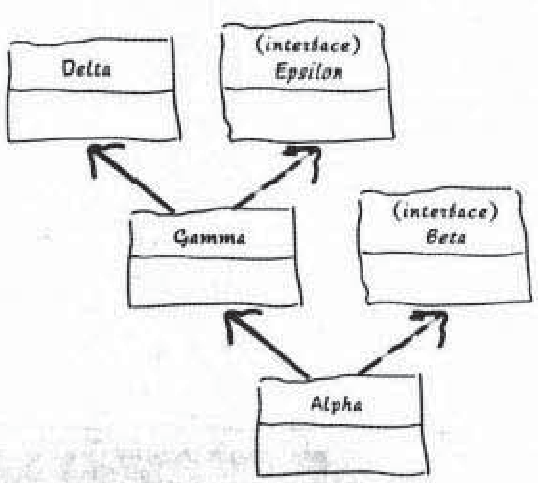
3)



4)



5)



Java 声明

2)

```
public abstract class Top { }
public class Tip extends Top { }
```

3)

```
public abstract class Fee { }
public abstract class Fi extends Fee { }
```

4)

```
public interface Foo { }
public class Bar implements Foo { }
public class Baz extends Bar { }
```

5)

```
public interface Zeta { }
public class Alpha implements Zeta { }
public interface Beta { }
public class Delta extends Alpha implements Beta { }
```



```
interface Nose {
  public int iMethod( ) ;
}
abstract class Picasso implements Nose {
  public int iMethod( ) {
    return 7;
  }
}
class Clowns extends Picasso { }

class Acts extends Picasso {
  public int iMethod( ) {
    return 5;
  }
}
```

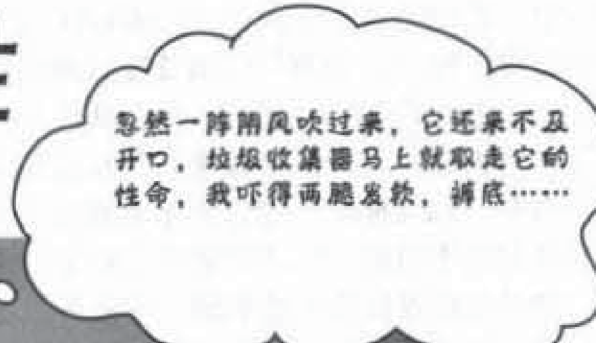
```
public class Of76 extends Clowns {
  public static void main(String [] args) {
    Nose [ ] i = new Nose [3] ;
    i[0] = new Acts( ) ;
    i[1] = new Clowns( ) ;
    i[2] = new Of76( ) ;
    for(int x = 0; x < 3; x++) {
      System.out.println( i [x] . iMethod( )
               + " " + i [x].getClass( ) );
    }
  }
}
```

输出

```
File Edit Window Help KillTheMime
%java Of76
5 class Acts
7 class Clowns
7 class  Of76
```

9 构造器与垃圾收集器

对象的前世今生





对象有生有死。 你必须为对象的生命循环周期负责。你决定着对象何时创建，如何创建，也决定着何时销毁对象。其实你不是真的自消灭对象，只是声明要放弃它而已。一旦它被放弃了，冷血无情的垃圾收集器（GC）就会将它蒸发掉、回收对象所占用的内存空间。如果你要编写Java程序，就必须创建对象。早晚你得将它们释放掉，不然就会出现内存不足的问题。这一章会讨论对象如何创建、存在于何处以及如何让保存和抛弃更有效率。这代表我们会述及堆、栈、范围、构造器、超级构造器、空引用等。注意：内容含有死亡成份，12岁以下儿童需由家长陪同观赏。

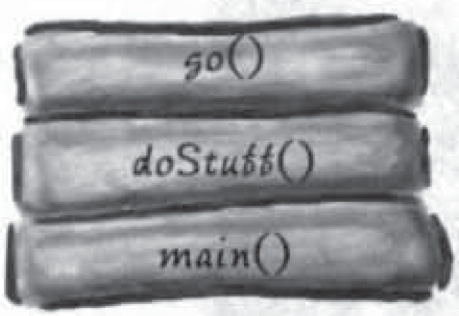
栈与堆：生存空间

在我们能够了解创建对象真正发生的情况之前，我们必须先退回一步来看。我们需要对生存在Java中的事物更加了解。这代表我们必须对栈与堆有更多的认识。在Java中，程序员会在乎内存中的两种区域：对象的生存空间堆（heap）和方法调用及变量的生存空间（stack）。当Java虚拟机启动时，它会从底层的操作系统取得一块内存，并以此区段来执行Java程序。至于有多少内存，以及你是否能够调整它都要看Java虚拟机与平台的版本而定。但通常你对这些事情无法加以控制。如果程序设计得不错的话，你或许也不太需要在乎。

我们知道所有的对象都存活于可垃圾回收的堆上，但我们还没看过变量的生存空间。而变量存在于哪一个空间要看它是哪一种变量而定。这里说的“哪一种”不是它的类型，而是实例变量或局部变量。后者这种区域变量又被称为栈变量，该名称已经说明了它所存在的区域。

栈

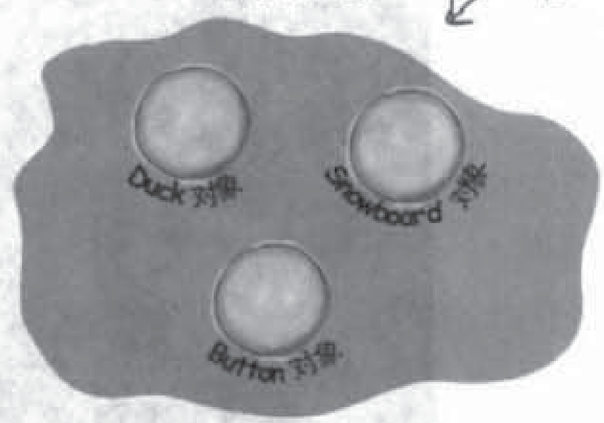
方法调用和局部变量。



堆

所有的对象

又称为可垃圾回收的堆



实例变量

实例变量是被声明在类而不是方法里面。它们代表每个独立对象的“字段”（每个实例都能有不同的值）。实例变量存在于所属的对象中。

```
public class Duck {
    int size;
}
```

每个Duck对象都会有独立的size

局部变量

局面变量和方法的参数都是被声明在方法中。它们是暂时的，且生命周期只限于方法被放在栈上的这段期间（也就是方法调用至执行完毕为止）。

```
public void foo(int x) {
    int i = x + 3;
    boolean b = true;
}
```

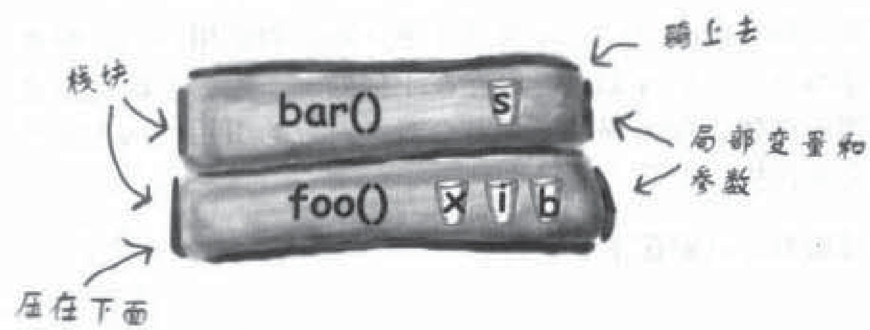
参数x和变量i、b都是局部变量

方法会被堆在一起

当你调用一个方法时，该方法会放在调用栈的栈顶。实际被堆上栈的是堆栈块，它带有方法的状态，包括执行到哪一行程序以及所有的局部变量的值。

栈顶上的方法是目前正在执行的方法（先假设只有一个，第14章有更多的说明）。方法会一直待在这里直到执行完毕，如果foo()方法调用bar()方法则bar()方法会放在 foo()方法的上面。

放了两个方法的栈



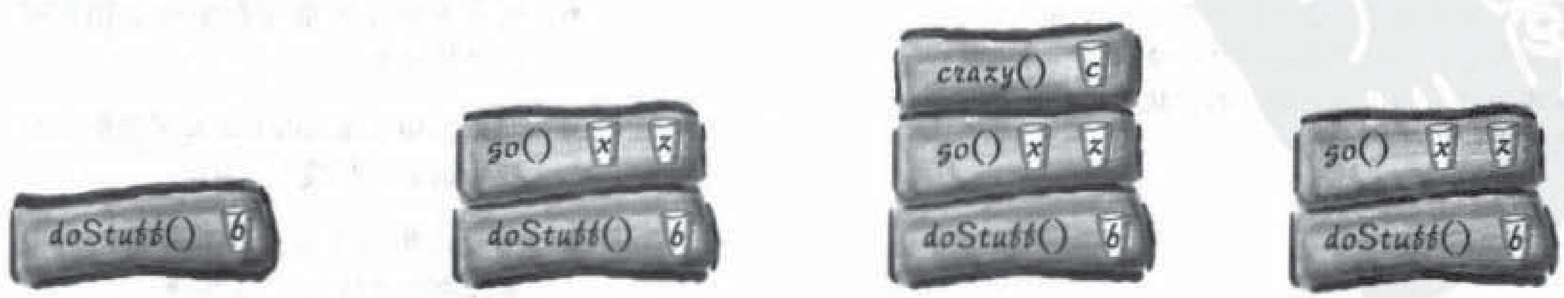
栈顶上方法是目前正在执行中的

```
public void doStuff() {
   boolean b = true;
   go(4);
}
public void go(int x) {
   int z = x + 24;
   crazy();
   // 假设还有很多程序代码
}
public void crazy() {
   char c = 'a';
}
```

stack的情境

左边有3个方法，第一个方法在执行过程中会调用第二个方法，第二个会调用第三个。每个方法都在内容中声明一个局部变量，而go()方法还有声明一个参数（这代表go()方法有两个局部变量）。

1. 某段程序代码调用了doStuff()使得doStuff()被放在stack最上方的栈块 中。
2. doStuff()调用go()。go()就被放在栈顶。
3. go()又调用crazy()使得crazy()现在处于栈顶。
4. 当crazy()执行完成后，它的堆栈块就被释放掉。执行就回到了go()。



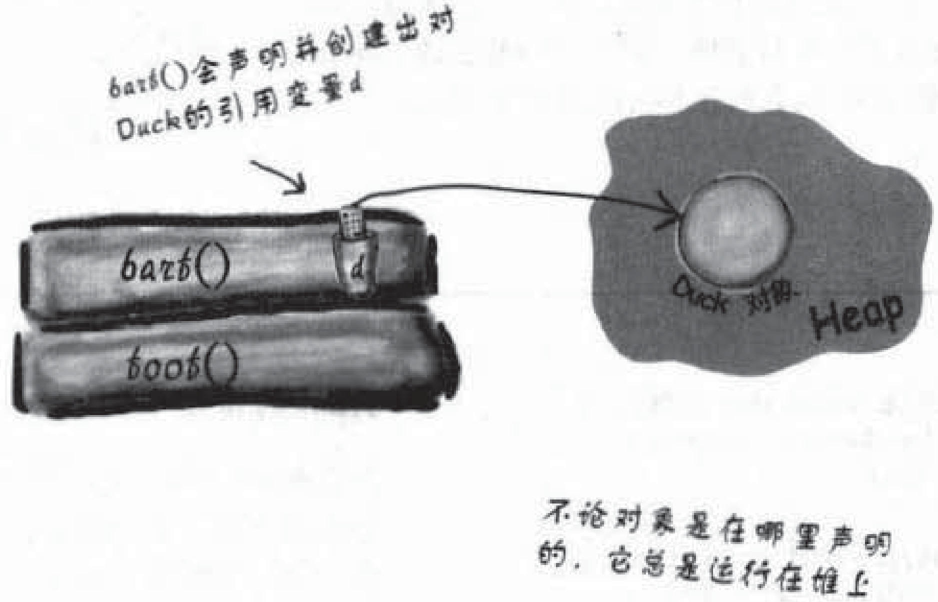
有关对象局部变量

要记得非primitive的变量只是保存对象的引用而已，而不是对象本身。你已经知道对象存在于何处——堆。不论对象是否声明或创建，如果局部变量是个对该对象的引用，只有变量本身会放在栈上。

对象本身只会存在于堆上。

```
public class StackRef {
   public void foof() {
      barf();
   }

   public void barf() {
      Duck d = new Duck(24);
   }
}
```



there are no Dumb Questions

问： **到底为什么要知道栈与堆的机制？这真地跟我有关系吗？**

答： 如果想要了解变量的有效范围（scope）、对象的建立、内存管理、线程（thread）和异常处理，则认识栈与堆是很重要的。后面的章节会讨论到有关线程与异常处理的部分，其余的都会在这一章讨论。你无需知道它们是如何实现的，只要能够理解这几页的内容就足够了。

要点

- 我们关心栈与堆这两种内存空间。
- 实例变量是声明在类中方法之外的地方。
- 局部变量声明在方法或方法的参数上。
- 所有局部变量都存在于栈上相对应的堆栈块中。
- 对象引用变量与primitive主数据类型变量都是放在栈上。
- 不管是实例变量或局部变量，对象本身都会在堆上。

如果局部变量生存在栈上，那么实例变量呢？

当你要新建一个CellPhone()时，Java必须在堆上帮CellPhone找一个位置。这会需要多少空间呢？足以存放该对象所有实例变量的空间。没错，实例变量存在于对象所属的堆空间上。

记住对象的实例变量的值是存放于该对象中。如果实例变量全都是primitive主数据类型的，则Java会依据primitive主数据类型的大小为该实例变量留下空间。int需要32位，long需要64位，依此类推。Java并不在乎私有变量的值，不管是32或 32,000,000的int都会占用32位。

但若实例变量是个对象呢？如果CellPhone对象带有一个Antenna对象呢？也就是说CellPhone带有Antenna类型的引用变量呢？

当一个新建对象带有对象引用的变量时，此时真正的问题是：是否需要保留对象带有的所有对象的空间？不是这样的。无论如何，Java会留下空间给实例变量的值。但是引用变量的值并不是对象本身，所以若CellPhone带有Antenna，Java只会留下Antenna引用量而不是对象本身所用到的空间。

那么Antenna对象会取得在堆上的空间吗？我们得先知道Antenna对象是在何时创建的。这要看实例变量是如何声明的。如果有声明变量但没有给它赋值，则只会留下变量的空间：

```
private Antenna ant;
```

直到引用变量被赋值一个新的Antenna对象才会在堆上占有空间：

```
private Antenna ant = new Antenna( );
```

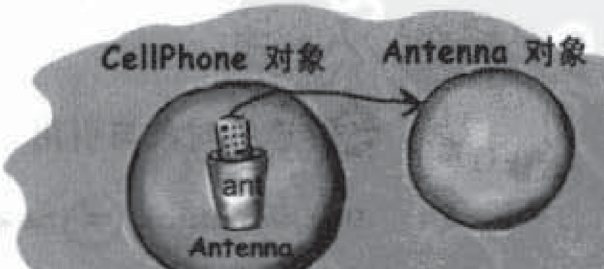


带有两个primitive主数据类型的实例变量的对象。变量所需的空间是在对象中。



对象带有引用到Antenna对象的变量，但是实际上没有初始Antenna对象的情形。

```
public class CellPhone {
  private Antenna ant;
}
```



对象带有一个新建出的Antenna对象

```
public class CellPhone {
  private Antenna ant = new Antenna();
}
```

创建对象的奇迹

现在你已经知道变量和对象的生存空间，我们可以开始更深入对象的创建。要记得声明对象和赋值有3个步骤：声明引用变量、创建对象、连接对象和引用。

但是第二个步骤还是个谜团——新对象的诞生。准备好接收新知识。

3个步骤的回顾：声明、创建、赋值

❶ 声明引用变量

创建出新的引用变量给该类型

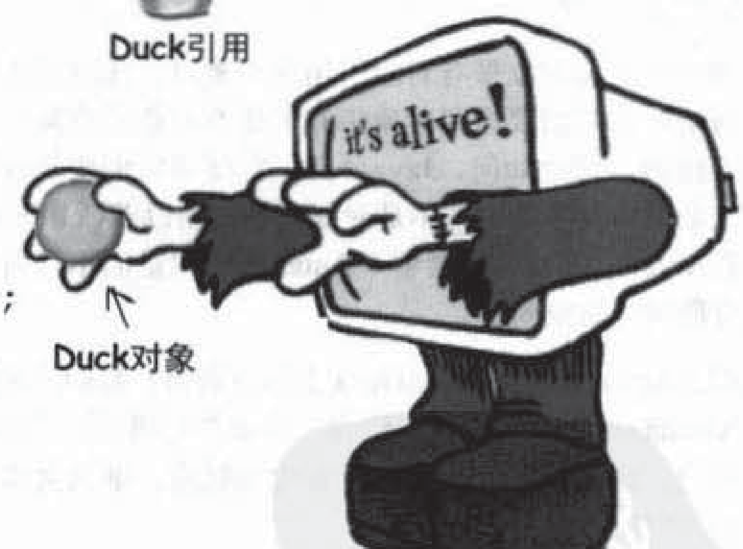
```
Duck myDuck = new Duck();
```



❷ 创建对象

呈现奇迹

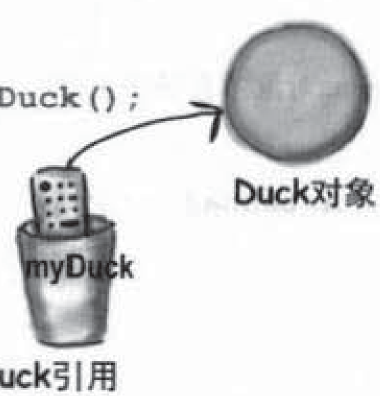
```
Duck myDuck = new Duck();
```



❸ 连接对象与引用

赋值对象给引用

```
Duck myDuck = new Duck();
```



看起来很像是在调用Duck()这个方法

看起来像是在调用Duck()方法

```
Duck myDuck = new Duck();
```

并不是。

我们是在调用 Duck 的构造函数。

构造函数看起很像方法，感觉上也很像方法，但它并不是方法。它带有new的时候会执行的程序代码。换句话说，这段程序代码会在你初始一个对象的时候执行。

唯一能够调用构造函数的办法就是新建一个类。（严格说起来，这是唯一在构造函数之外能够调用构造函数的方式，本章稍后会讨论这个部分）。

构造函数带有你在初始化对象时会执行的程序代码。也就是新建一个对象时就会被执行。

就算你没有自己写构造函数，编译器也会帮你写一个。

哪里来的构造函数？

我们没有写啊，难道说这本书有缺页？

你可以帮类编写构造函数，但如果你没有写，编译器会偷偷帮你写！

下面就是编译器写出来的

```
public  Duck() {

}
```

有没有发现少了什么？这跟方法有什么不同之处？

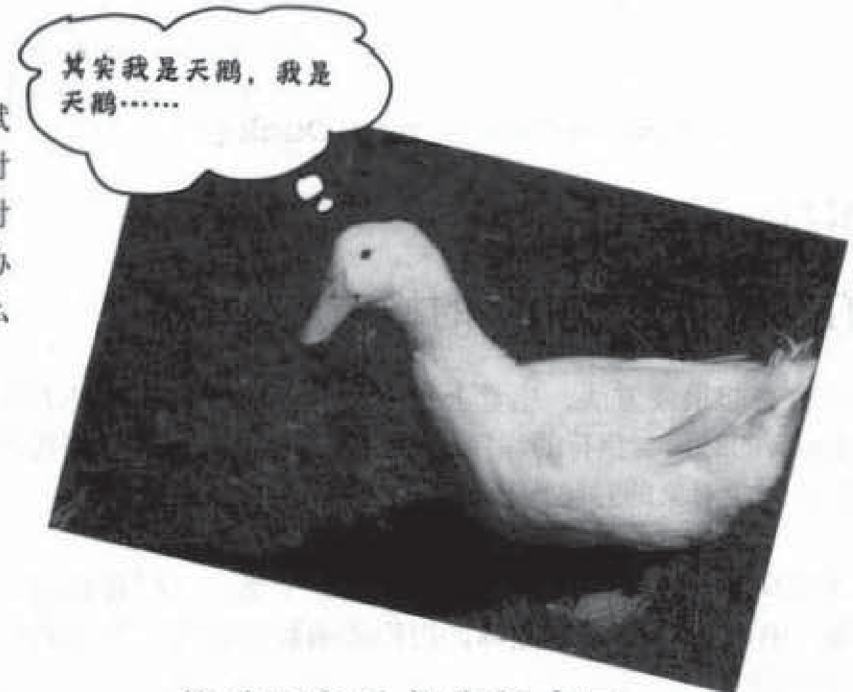
一定要与类的名称相同

```
public  Duck() {
   // 构造代码在此
}
```

方法有返回类型，构造函数没有返回类型

构造Duck

构造函数的一项关键特征是它会在对象能够被赋值给引用之前就执行。这代表你可以有机会在对象被使用之前介入。也就是说，在任何人取得对象的遥控器前，对象有机会对构造过程给予协助。在Duck的构造函数中，我们没有作出什么有意义的事情，但还是有展示出事件的顺序。




```
public class Duck {

   public Duck() {
      System.out.println("Quack");
   }
}
```

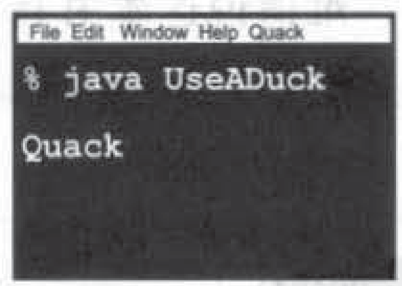
列出一行聒叫

构造函数让你有机会可以介入new的过程

```
public class UseADuck {

   public static void main (String[] args) {
      Duck d = new Duck();
   }
}
```

这样会启动Duck的构造函数



Sharpen your pencil

构造函数让你可以在构造过程的步骤中参一脚。你能否想象这有什么用处吗？如果Car是赛车的类，对于右边几个情境你是否能想象出构造函数的用途？可以的话就打个勾。

- ❑ 记录已经构造出多少部赛车。
- ❑ 记录特定的状态。
- ❑ 给实例变量赋值。
- ❑ 留下创建对象的证据。
- ❑ 将对象加到ArrayList中。
- ❑ 创建HAS-A对象。
- ❑ ____________________ （自己想）

新建Duck状态的初始化

大部分的人都是使用构造函数来初始化对象的状态。也就是说设置和给对象的实例变量赋值。

```
public Duck() {
    size = 34;
}
```

这在开发者知道Duck类应该有多大时是没问题的。但如果是要由使用Duck的程序员来决定时应该怎么办？

你可以使用该类的setSize()来设定大小。但这会让Duck暂时处于没有大小数值的状态（实例变量没有默认值），且需要两行才能搞定。下面就是这么做的：

```
public class Duck {
    int size;  ← 实例变量

    public Duck() {
        System.out.println("Quack");  ← 构造函数
    }

    public void setSize(int newSize) {  ← setter方法
        size = newSize;
    }
}
```

```
public class UseADuck {

    public static void main (String[] args){
        Duck d = new Duck();
                 ←
        d.setSize(42);
    }
}
```

问题出在这里，Duck在此处已经建立，但是却没有size值！你必须依赖Duck的用户记得要设定大小。

there are no Dumb Questions

问： 既然编译器会帮你写，那为何还要自己写构造函数？

答： 如果你在创建对象时需要有程序代码帮忙初始化，那你就得自己编写构造函数。例如说你需要通过用户的输入来完成对象的创建。另外一个原因与父类的构造函数有关，稍后会讨论这个部分。

问： 如何分辨构造函数和方法？

答： Java可以有与类同名的方法而不会变成构造函数。其中的差别在于是否有返回类型。构造函数不会有返回类型。

问： 构造函数会被继承吗？

答： 不会。我们稍后会讨论到这个部分。

使用构造函数来初始化Duck的状态



如果某种对象不应该在状态被初始化之前就使用，就别让任何人能够在没有初始化的情况下取得该种对象！让用户先构造出Duck对象再来设定大小是很危险的。如果用户不知道，或者忘记要执行setSize()怎么办？

最好的方法是把初始化的程序代码放在构造函数中，然后把构造函数设定成需要参数的。

```
public class Duck {
   int size;

   public Duck(int duckSize) {     // 给构造函数加上参数
      System.out.println("Quack");

      size = duckSize;             // 使用参数的值来设定size这个实例变量

      System.out.println("size is " + size);
   }
}
```

```
public class UseADuck {

  public static void main (String[] args) {
     Duck d = new Duck(42);        // 传值给构造函数
  }
}
```

只用一行就可以创建出新的Duck并且设定好大小

```
File Edit Window Help Honk
% java UseADuck
Quack
size is 42
```

Duck的简易饲养方法

一定要有不需参数的构造函数。

如果Duck的构造函数需要一项参数会怎样？想想看。上一页的Duck只有一个构造函数，且它需要一个int型的size参数。这也许不是个问题，但却让程序员感到更为困难，特别是在不知道Duck的大小时。如果有预设的大小让程序员在不知道适当大小时也可以创建出Duck不是更好吗？

想象一下你可以让用户在创建Duck时有两个选项：一个可以指定Duck的大小（通过构造函数的参数），另外一个使用默认值而无需指定大小。

你无法只依靠单一的构造函数就能够很清楚地达到这个目的。要记得，如果某个方法或构造函数有一项参数，你就必须在调用该方法或构造函数的时候传入适当的参数。你没有办法作出一种没给参数时就使用默认值的方法，因为在这个情况下没有给参数就无法通过编译程序。也许你可以用下面这种不太理想的方法取代：

```
public class Duck {
   int size;

   public Duck(int newSize) {
      if (newSize == 0) {
         size = 27;
      } else {
         size = newSize;
      }
   }
}
```

如果参数值为0就使用默认的大小。

这代表程序员必须要知道传入0对于创建Duck的构造函数意味着要使用默认的大小而不是真正的0。万一程序员真的做出0大小的Duck怎么办？这样的问题在于传入0的意图无法确实的分辨。

你需要有两种方法来创建出新的Duck：

```
public class Duck2 {
   int size;

   public Duck2() {
      // 指定默认值
      size = 27;
   }

   public Duck2(int duckSize) {
      // 使用参数设定
      size = duckSize;
   }
}
```

知道大小时：

```
Duck2 d = new Duck2(15);
```

不知道大小时：

```
Duck2 d2 = new Duck2();
```

因此这会需要两个构造函数来分辨两种选项。一个需要参数，另外一个不需要参数。如果一个类有一个以上的构造函数，这代表它们也是重载的。

编译器一定会帮你写出没有参数的构造函数吗？ No！

你可能会认为如果你只编写有参数的构造函数，编译器会看出你没有无参数的构造函数而帮你弄出一个来。别再相信没有事实根据的说法了，编译器只会在你完全没有设定构造函数时才会调用。

如果你已经写了一个有参数的构造函数，并且你需要一个没有参数的构造函数，则你必须自己动手写！

只要你有自己写的构造函数，不管是哪一种，这都会像是在跟编译器说："老兄，我自己的构造函数不用你管"。

如果类有一个以上的构造函数，则参数一定要不一样。

这包括了参数的顺序与类型，只要是不一样就可以。这就跟方法的重载是相同的，不过细节会留到其他的章节再讨论。



重载构造函数的意思代表你有一个以上的构造函数且参数都不相同

下面列出的构造函数都是合法的，因为参数都不相同。假设说有两个构造函数的参数都是只有一个int，则肯定无法通过编译程序。编译器看的是参数的类型和顺序而不是参数的名字。你可以做出相同类型但是顺序不同的参数。使用String以及int型的参数顺序与使用int以及String型的参数顺序是不同的。

4个不同的构造函数代表有4种不同的创建方式



```
public class Mushroom {

  public Mushroom(int size) { }

  public Mushroom( ) { }

  public Mushroom(boolean isMagic) { }

  public Mushroom(boolean isMagic, int size) { }

  public Mushroom(int size, boolean isMagic) { }

}
```

你知道要做多大

什么都不知道

知道是否为蘑菇，但不知道大小

知道大小与是否有魔力

因为顺序不同所以可以过关

要点

- 实例变量保存在所属的对象中，位于堆上。
- 如果实例变量是个对对象的引用，则引用与对象都是在堆上。
- 构造函数是个会在新建对象的时候执行程序代码。
- 构造函数必须与类同名且没有返回类型。
- 你可以用构造函数来初始被创建对象的状态。
- 如果你没有写构造函数，编译器会帮你安排一个。
- 默认的构造函数是没有参数的。
- 如果你写了构造函数，则编译器就不会调用。
- 最好能有无参数的构造函数让人可以选择使用默认值。
- 重载的构造函数意思是有超过一个以上的构造函数。
- 重载的构造函数必须有不同的参数。
- 两个构造函数的参数必须不同。
- 实例变量有默认值，原始的默认值是 0/0.0/false，引用的默认值是 null。

Sharpen your pencil

猜猜看哪个新建Duck()会用到哪个构造函数？我们已经帮你找到一个。

```
public class TestDuck {

  public static void main(String[] args){

    int weight = 8;
    float density = 2.3F;
    String name =  "Donald" ;
    long[] feathers = {1,2,3,4,5,6};
    boolean canFly = true;
    int airspeed = 22;

    Duck[] d = new Duck[7];

    d[0] = new Duck();

    d[1] = new Duck(density, weight);

    d[2] = new Duck(name, feathers);

    d[3] = new Duck(canFly);

    d[4] = new Duck(3.3F, airspeed);

    d[5] = new Duck(false);

    d[6] = new Duck(airspeed, density);
  }
}
```

```
class Duck {

  int pounds = 6;
  float floatability = 2.1F;
  String name = "Generic";
  long[] feathers = {1,2,3,4,5,6,7};
  boolean canFly = true;
  int maxSpeed = 25;

  public Duck() {
    System.out.println("type 1 duck");
  }

  public Duck(boolean fly) {
    canFly = fly;
    System.out.println("type 2 duck");
  }

  public Duck(String n, long[] f) {
    name = n;
    feathers = f;
    System.out.println("type 3 duck");
  }

  public Duck(int w, float f) {
    pounds = w;
    floatability = f;
    System.out.println("type 4 duck");
  }

  public Duck(float density, int max) {
    floatability = density;
    maxSpeed = max;
    System.out.println("type 5 duck");
  }
}
```

问： **先前你说过最好要有没参数的构造函数以便让用户可以调用使用我们提供默认值的构造函数。但若不可能提供默认值的时候还应该要提供无参数的构造函数吗？**

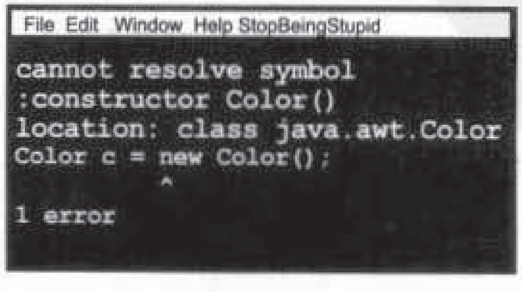
答： 没错。有时候有默认值的无参数构造函数是不合理的。在Java API中有些类就没有无参数的构造函数，例如说Color这个类。它是用来设定字型或GUI按钮的颜色。当你要制作Color的实例时，该实例会代表特定的颜色。如果你要用到Color，就必须以某种方式指定颜色：

```
Color c = new Color(3, 45, 200);
```

这是使用3个int来代表RGB三色的构造函数，后面讨论Swing的章节会有说明。如果没有给颜色，那Java API的设计人也许可以给你一个预设桃红色，想要吗？如果以这种方式创建Color对象：

```
Color c = new Color( );
```

编译器会向你抱怨没有这样的构造函数：



构造函数迷你小回顾：

- 构造函数是在新建类时会执行的程序。

```
Duck d = new Duck();
```

- 构造函数必须与类的名字一样，且没有返回类型。

```
public Duck(int size) { }
```

- 如果你没有写构造函数，则编译器会帮你写一个没有参数的

```
public Duck() { }
```

- 一个类可以有很多个构造函数，但不能有相同的参数类型和顺序，这叫作重载过的构造函数。

```
public Duck() { }

public Duck(int size) { }

public Duck(String name) { }

public Duck(String name, int size) { }
```

BRAIN POWER

想到父类吗？

在创建Dog的时候，Canine的构造函数是否应该要执行？

如果父类是抽象的，那它可以有构造函数吗？

接下来会看到这些主题，你现在应该独立地思考一下构造函数与父类之间的关系。

there are no Dumb Questions

问： **构造函数应该是公有的吗？**

答： 不。构造函数可以是公有、私有或不指定的。第16章和附录B有关于不指定的预设存取权讨论。

问： **一个私有的构造函数有什么作用？没有人能够调用它，所以也就没有人能够创建该对象？**

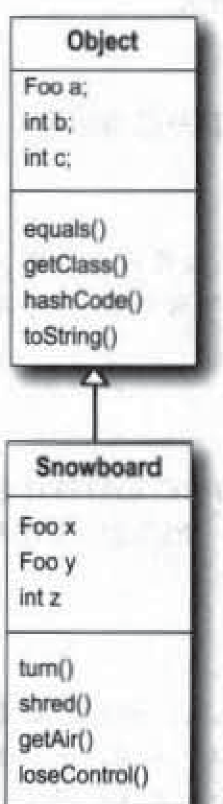
答： 不是这么说的。私有不是完全不能存取，它代表该类以外不能存取。这听起来很矛盾吧？下一章会讨论这个问题。

研究显示挑战智力大考验最多可以提升神经的大小到42%以上。

等一下……我们都还没有谈到父类以及继承与构造函数之间的关系

这样才有趣。还记得上一章我们说到Snowboard对象把Object部分包在自己的核心吗？那个讨论的重点在于每个对象不只是保存自行声明的变量，还有从父类来的所有东西（至少会带有Object，因为每个类都有继承过这个类）。

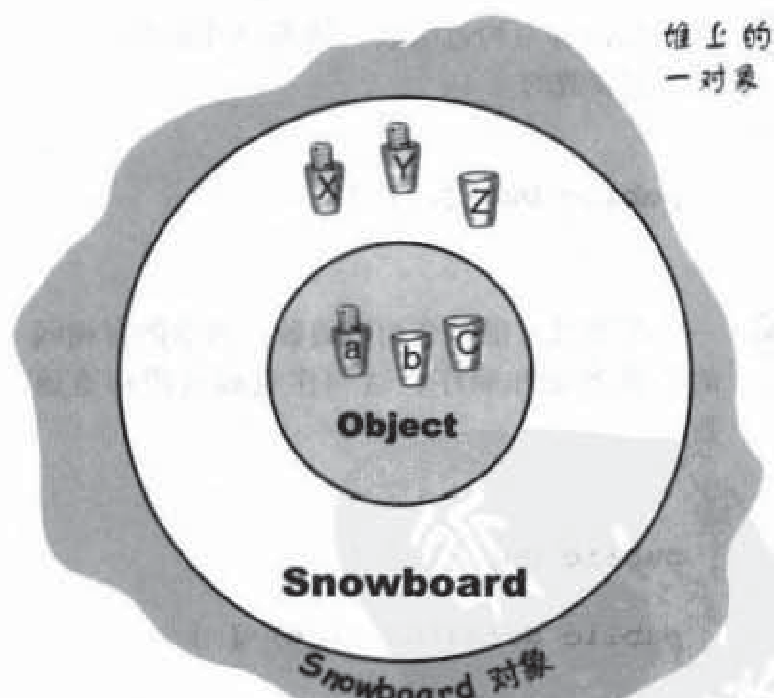
因此在创建某个对象时（完全没有其他方法能够创建对象，只能通过new来产生新对象），对象会取得所有实例变量所需的空间，这当然也包括一路继承下来的东西。想象一下，父类也许会有一个setter以包装私用的变量。但此变量必须有空间。概念上，你可以把整个情境用下面的图来表示，对象就像洋葱是有层次的（请见《史瑞克》），每一层都代表某一级的父类。



对象带有被存取方法所包装的实例变量。这些实例变量是在子类被初始化的时候创建的（它们不算真正的对象，但因为它们有被包装过，所以我们也不在乎）。

Snowboard也有自己的实例变量，因此Snowboard需要自己加上Object的空间。

堆上的单一对象



此图所示只有一个对象。但因为它带有Snowboard与Object的内容，所以这两个类的所有实例变量也会在这里。

父类的构造函数在对象的生命中所扮演的角色

在创建新对象时，所有继承下来的构造函数都会执行。

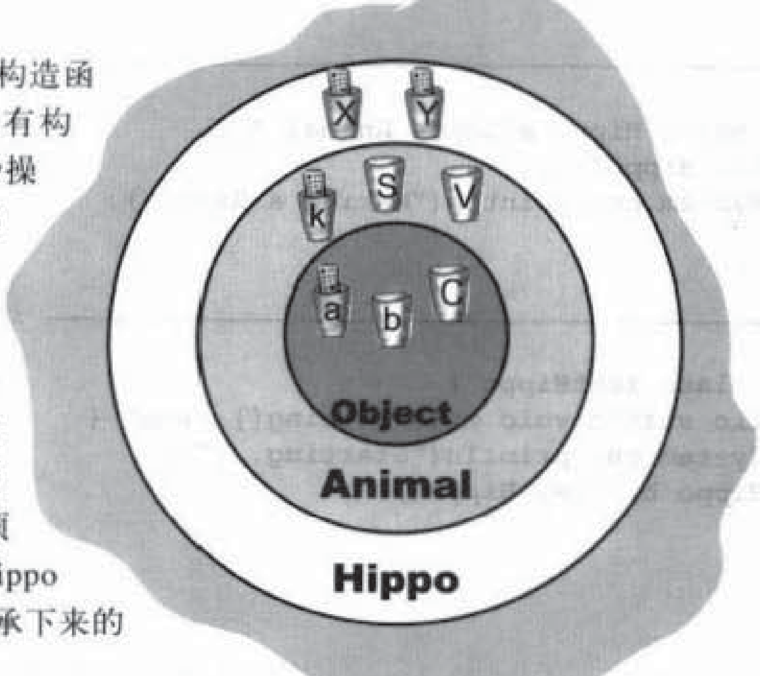
这代表每个父类都有一个构造函数（因为每个类至少都会有一个构造函数），且每个构造函数都会在子类对象创建时期执行。

执行new的指令是个重大事件，它会启动构造函数连锁反应。还有，就算是抽象的类也有构造函数。虽然你不能对抽象的类执行new操作，但抽象的类还是父类，因此它的构造函数会在具体子类创建出实例时执行。

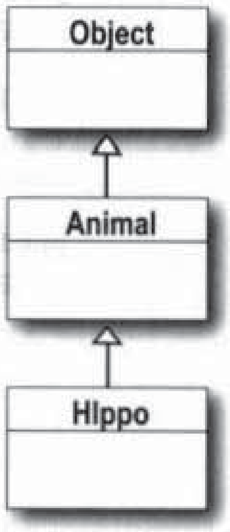
在构造函数中用super调用父类的构造函数的部分。要记得子类可能会根据父类的状态来继承方法（也就是父类的实例变量）。完整的对象需要也是完整的父类核心，所以这就是为什么父类构造函数必须执行的原因。就算Animal上有些变量是Hippo不会用到的，但Hippo 可能会用到某些继承下来的方法必须读取Animal的实例变量。

构造函数在执行的时候，第一件事是去执行它的父类的构造函数，这会连锁反应到Object这个类为止。

我们在接下来的几页会看到父类构造函数是如何被调用，以及你如何自行调用它们。你也会知道要如何调用有参数的父类构造函数。



在堆上的Hippo对象



Hippo对象IS-A Animal同时也IS-A Object。如果你要创建出Hippo，也得创建出Animal与Object的部分。

这样的过程被称为“构造函数链（Constructor Chaining）”

创建Hippo也代表创建Animal与Object

```java
public class Animal {
   public Animal() {
      System.out.println("Making an Animal");
   }
}
```

```java
public class Hippo extends Animal {
   public Hippo() {
      System.out.println("Making a Hippo");
   }
}
```

```java
public class TestHippo {
   public static void main (String[] args) {
      System.out.println("Starting...");
      Hippo h = new Hippo();
   }
}
```

Sharpen your pencil

哪一个输出才是对的？执行左边的程式代码得到A还是B的结果？

答案就在下面。

A

```
File Edit Window Help Swear
% java TestHippo
Starting...
Making an Animal
Making a Hippo
```

B

```
File Edit Window Help Swear
% java TestHippo
Starting...
Making a Hippo
Making an Animal
```

1. 某个程序执行new Hippo()的动作，Hippo()的构造函数进入堆栈最上方的堆栈块



2. Hippo()调用父类的构造函数导致Animal()的构造函数进入栈顶



3. Animal()调用父类的构造函数导致Object()的构造函数进入栈顶



4. Object()执行完毕，它的堆栈块被弹出，接着继续执行Animal()的



答案是 A，Hippo()是最先被调用的，但却是最后一个执行完毕。

如何调用父类的构造函数?

以Duck的构造函数来说，你也许认为它会直接调用Animal()，但实际上不是这样的：

```
public class Duck extends Animal {
    int size;

    public Duck(int newSize) {
错! →   Animal(); ← 这不合法!
        size = newSize;
    }
}
```

调用父类构造函数的唯一方法是调用super()。

它看起来会像下面这样：

```
public class Duck extends Animal {
    int size;

    public Duck(int newSize) {
        super(); ← 要这么调用
        size = newSize;
    }
}
```

在你的构造函数中调用super()会把父类的构造函数放在堆栈的最上方。你猜父类的构造函数会做什么？它会调用它的父类构造函数。这会一路上去直到Object的构造函数为止。然后再一路执行、弹出回到原来的构造函数。

如果我们没有调用super()会发生什么事?

你也许已经猜到了。

编译器会帮我们加上super()的调用。

所以编译器有两种涉入构造函数的方式：

● **如果你没有编写构造函数。**

```
public ClassName() {
    super();
}
```

● **如果你有构造函数但没有调用super()。**

编译器会帮你对每个重载版本的构造函数加上下面这种调用：

```
super();
```

编译器帮忙加的一定会是没有参数的版本，假使父类有多个重载版本，也只有无参数的这个版本会被调用到。

小孩能够在父母之前出生吗？

如果你把父类想象成子类的父母，那就可以看出来谁是先存在的。父类的部分必须在子类创建完成之前就必须完整地成型。记住，子类对象可能需要动用到从父类继承下来的东西，所以那些东西必须要先完成。父类的构造函数必须在子类的构造函数之前结束。



再看一下252页的堆栈，你会发现Hippo的构造函数是第一个被调用的（在堆栈上的第一个），却也是最后一个完成的！每个子类的构造函数会立即调用父类的构造函数，如此一路往上直到Object。等到Object完成后会回去执行Animal的，然后等Animal完成后又回去执行Hippo剩下的构造函数。这是因为：

对super()的调用必须是构造函数的第一个语句*。

类Boop的可能构造函数

```
R public Boop() {
      super();
  }

R public Boop(int i) {
      super();
      size = i;
  }
```

这么做是可以的，因为super()的调用是第一个命令

```
R public Boop() {
  }
```

这么做也可以，因为编译器会自动把super()加到最前面

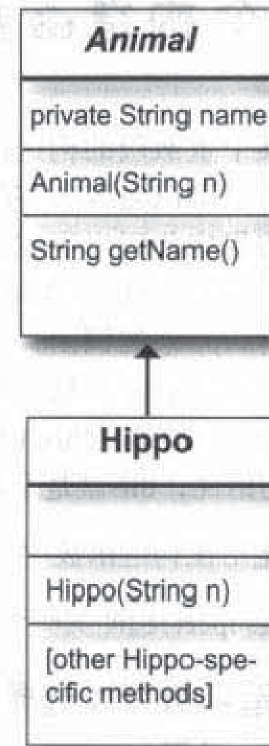
```
R public Boop(int i) {
      size = i;
  }

⊘ public Boop(int i) {
      size = i;
      super();
  }
```

这就不行了，编译器不会让你这么做

*有异常情况，见256页

有参数的父类构造函数



如果父类的构造函数有参数该怎么办？你能够传值进去吗？如果不行的话，则没有无参数构造函数的类将不能被继承。想象这个情景：所有的动物都有名字。所以Animal这个类有个getName()可以返回name实例变量的值。此实例变量是被标记为私有的，但Hippo这个子类有把getName()继承下来。问题来了：

Hippo有getName()这个方法但是没有name实例变量。Hippo要靠Animal的部分来维持name实例变量，然后从getName()来返回这个值，但Animal要如何取得这个值呢？唯一的机会是通过super()来引用父类，所以要从这里把name的值传进去，让Animal把它存到私有的name实例变量中。

```
public abstract class Animal {
   private String name;          ← 每个Animal都会有名字

   public String getName() {     ← Hippo会继承这个getter
      return name;
   }

   public Animal(String theName) {
      name = theName;            ← 有参数的构造函数，用来设定name
   }
}
```




```
public class Hippo extends Animal {

   public Hippo(String name) {
      super(name);               ← 这个构造函数会要求名称
   }                             ↖ 传给Animal的构造函数
}
```

```
public class MakeHippo {
   public static void main(String[] args) {
      Hippo h = new Hippo("Buffy");          ← 创建Hippo，传入名字然后
      System.out.println(h.getName());       ← 再列出来看看
   }
}
```

```
File Edit Window Help Hide
%java MakeHippo
Buffy
```

从某个构造函数调用重载版的另一个构造函数

如果有某个重载版的构造函数除了不能处理不同类型的参数之外，可以处理所有的工作，那要怎么办？你不想让相同的程序代码出现在每个构造函数中（维护起来很麻烦），所以你想把程序代码只摆在某个构造函数中（包括对super()的调用）。如此一来，所有的构造函数都会先调用该构造函数，让它来执行真正的构造函数。这很容易，只要调用this()或this(aString)或this(27, x) 就行。换句话说，this就是个对对象本身的引用。

this()只能用在构造函数中，且它必须是第一行语句！

这样会跟super()起冲突吗？所以你必须选择：

每个构造函数可以选择调用super()或this()，但不能同时调用！

使用this()来从某个构造函数调用同一个类的另外一个构造函数。

this()只能用在 构造函数中，且必须是第一行语句。

super()与this()不能兼得。

```
class Mini extends Car {

    Color color;

    public Mini() {
       this(Color.Red);
    }

    public Mini(Color c) {
       super("Mini");
       color = c;
       // 初始化动作
    }

    public Mini(int size) {
       this(Color.Red);
       super(size);

    }
}
```

无参数的构造函数以默认的颜色调用真正的构造函数

这才是真正的构造函数

有问题！不能同时调用super()和this()，两者只有一个会是第一行语句

```
File Edit Window Help Drive
javac Mini.java
Mini.java:16: call to super must
be first statement in constructor
      super();
        ^
```

Sharpen your pencil

下面类SonOfBoo的构造函数中有某几个无法通过编译，试试看你能否找到不合法的是哪几个。在有问题的构造函数旁边划条线连到右边的错误信息上。

```
public class Boo {
   public Boo(int i) { }
   public Boo(String s) { }
   public Boo(String s, int i) { }
}
```

```
class SonOfBoo extends Boo {

   public SonOfBoo() {
      super("boo");
   }

   public SonOfBoo(int i) {
      super("Fred");
   }

   public SonOfBoo(String s) {
      super(42);
   }

   public SonOfBoo(int i, String s) {
   }

   public SonOfBoo(String a, String b, String c) {
      super(a,b);
   }

   public SonOfBoo(int i, int j) {
      super("man", j);
   }

   public SonOfBoo(int i, int x, int y) {
      super(i, "star");
   }
}
```

Make it Stick



玫瑰是红的，慧星是无辜的。

先上车才能补票，先有父类才能有子类。

对象的父类部分必须先完备，子类部分才能创建，就像你父母要先出生才会轮到你出生（复制人吗？也得要先有来源吧）。

File Edit Window Help

```
%javac SonOfBoo.java
cannot resolve symbol
symbol : constructor Boo
(java.lang.String,java.
lang.String)
```

File Edit Window Help Yadayadayada

```
%javac SonOfBoo.java
cannot resolve symbol
symbol  : constructor Boo
(int,java.lang.String)
```

File Edit Window Help ImNotListening

```
%javac SonOfBoo.java
cannot resolve symbol
symbol:constructor Boo()
```

我们已经了解了对象的诞生过程，但对象会存活多久呢？

对象的生命周期完全要看引用到它的“引用”。如果引用还活着，则对象也会继续活在堆上。如果引用死了（稍后会解释），则对象就会跟着殉情……陪葬……送命……

如果对象生命周期要看引用变量的生命周期而定，那变量到底会活多久？

这又要看它是局部变量或实例变量而定。下面的程序展示出局部变量的生命周期。

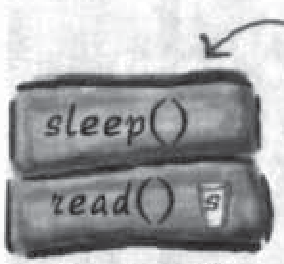
```
public class TestLifeOne {

    public void read() {
        int s = 42;
        sleep();
    }

    public void sleep() {
        s = 7;
    }
}
```

s的范围只限于read()里面，别处无法使用

非法使用！



sleep()无法存取s变量，因为它不在sleep()的堆栈块中

s还活着，但范围仅限于read()，当sleep()执行完毕回到read()时，read()还是能使用s，但当read()执行完毕被弹出时，家属就得帮s举办追悼会。

① **局部变量只会存活在声明该变量的方法中**

```
public void read() {
    int s = 42;
    // 's' 只能用在此方法中
    // 当方法结束时
    // s会完全消失
```

变量s只能用在read()方法中。换句话说，此变量的范围只会在所属方法的范围内。其余的程序代码完全见不到s。

② **实例变量的寿命与对象相同。如果对象还活着，则实例变量也会是活的。**

```
public class Life {
    int size;

    public void setSize(int s) {
        size = s;
        // 's' 会在方法结束时
        // 消失，但size在类中
        // 到处都可用

    }
```

此时s变量（这次是方法的参数）的范围同样也只限制在所属的setSize()这个方法中。

“life”与“scope”的差别

```
public void doStuff() {
   boolean b = true;
   go(4);
}

public void go(int x) {
   int z = x + 24;
   crazy();
   // 这里有更多的代码
}

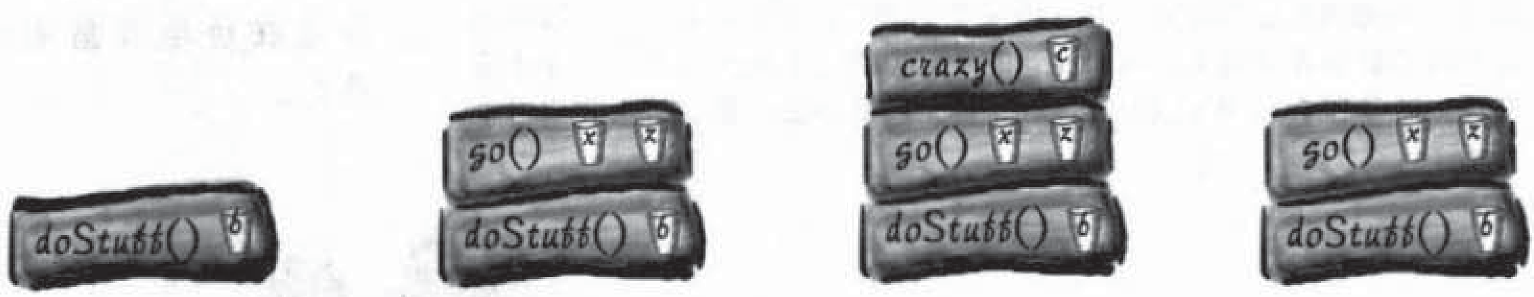
public void crazy() {
   char c = 'a';
}
```

Life

只要变量的堆栈块还存在于堆栈上，局部变量就算活着。也就是说，活到方法执行完毕为止。

Scope

局部变量的范围只限于声明它的方法之内。当此方法调用别的方法时，该变量还活着，但不在目前的范围内。执行其他方法完毕返回时，范围也就跟着回来。



1. doStuff()运行在堆栈，变量b存活于scope中。
2. 调用go()，x，z，b都活着，但只有b不在它的范围中。
3. 调用crazy()，只有c在它的范围中。
4. 完成 crazy()，c既不在它的范围中，也没活下来，只有x和z在它的范围中。

当局部变量活着的时候，它的状态会被保存。只要doStuff()还在堆栈上，b变量就会保持它的值。但b变量只能在doStuff()待在栈顶时才能使用。也就是说局部变量只能在声明它的方法在执行中才能被使用。

那引用变量呢？

引用的规则与primitive主数据类型相同。引用变量只能在处于它的范围中才能被引用，也就是说除非引用变量是在它的范围中，不然就不能使用对象的遥控器。真正的问题是：

“变量的生命周期如何影响对象的生命周期？”

只要有活着的引用，对象也就会活着。如果某个对象的引用已经不在它的范围中，但此引用还是活着的，则此对象就会继续活在堆上。

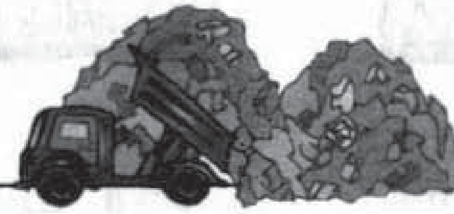
如果对对象的唯一引用死了，对象就会从堆中被踢开。引用变量会跟堆栈块一起解散，因此被踢开的对象也就正式的声明出局。关键在于知道何时对象会变成可被垃圾收集器回收的。

一旦对象符合垃圾收集器（GC）的条件，你就无需担心回收内存的问题。如果程序内存不足，GC就会去歼灭部分或全部的可回收对象。你可能还是会遇到内存不足的状况，但这要等到所有可回收的都被回收掉也还不够的时候才会发生。你要注意的是对象用完了就要抛弃，这样才能让垃圾收集器有东西可以回收。如果你把持着对象不放，垃圾收集器也帮不了什么忙。

除非有对对象的引用，否则该对象一点意义也没有。

如果你无法取得对象的引用，则此对象只是浪费空间罢了。

但若对象是无法取得的，GC会知道该怎么做。那种对象迟早会葬送在垃圾收集器的手上。



当最后一个引用消失时，对象就会变成可回收的。

有3种方法可以释放对象的引用：

① 引用永久性的离开它的范围。

```
void go() {
  Life z = new Life();
}
```

z会在方法结束时消失

② 引用被赋值到其他的对象上。

```
Life z = new Life();
z = new Life();
```

第一个对象会在z被赋值到别处时挂掉

③ 直接将引用设定为null。

```
Life z = new Life();
z = null;
```

第一个对象会在z被赋值为null时击毙

对象杀手一号

引用永久性的离开它的范围



```
public class StackRef  {
   public void foof() {
      barf();
   }

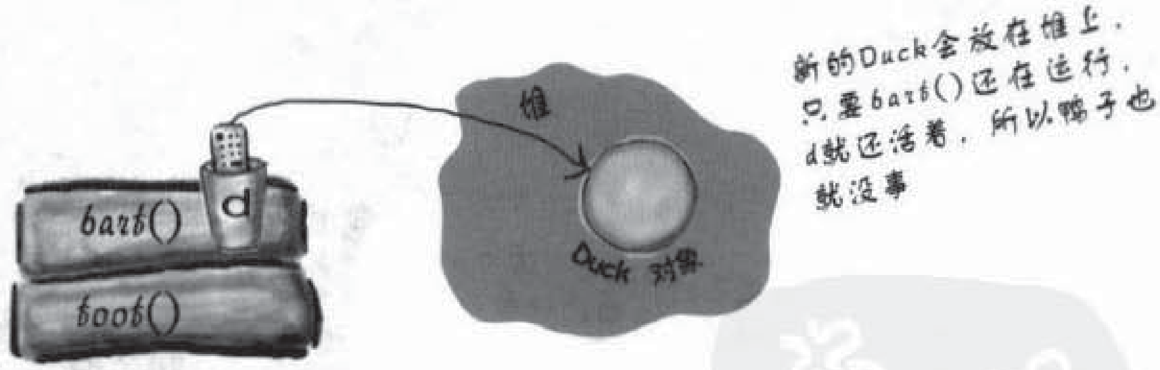
   public void barf() {
      Duck d = new Duck();
   }
}
```



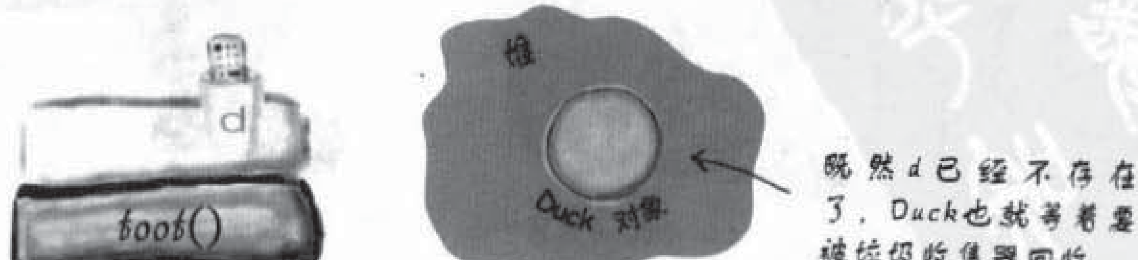
❶ foof()被推到堆栈上，没有声明变量



❷ barf()被推到堆栈上，创建一个对象以及对它的引用

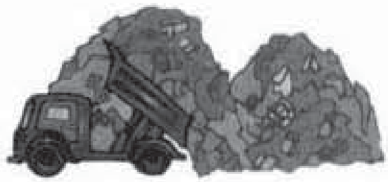


❸ barf()执行完毕，因此d也就挂了



对象杀手二号

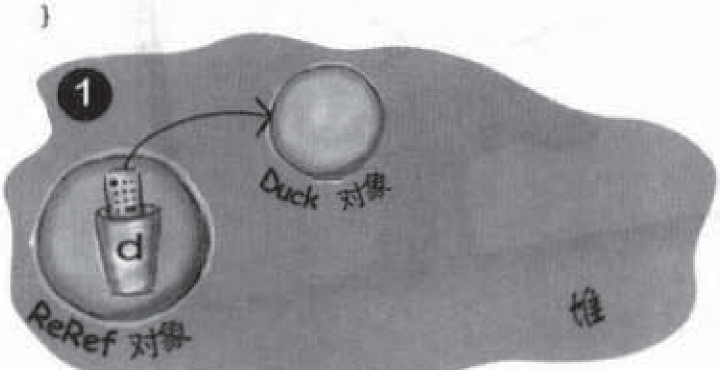
引用被赋值到其他的对象上



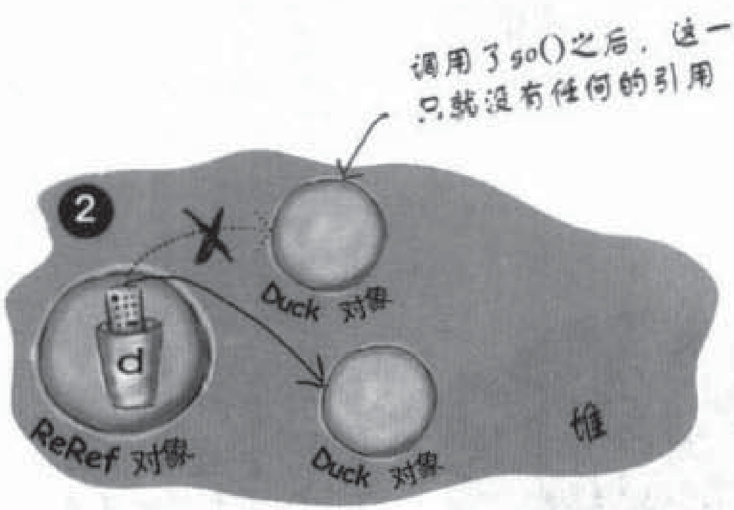
```
public class ReRef {

    Duck d = new Duck();

    public void go() {
      d = new Duck();
    }
}
```



新的Duck会放在堆上，只要ReRef还活着，d就没事，除非……



既然d引用到其他的Duck，第一个Duck就跟死掉是一样的



物件杀手三号

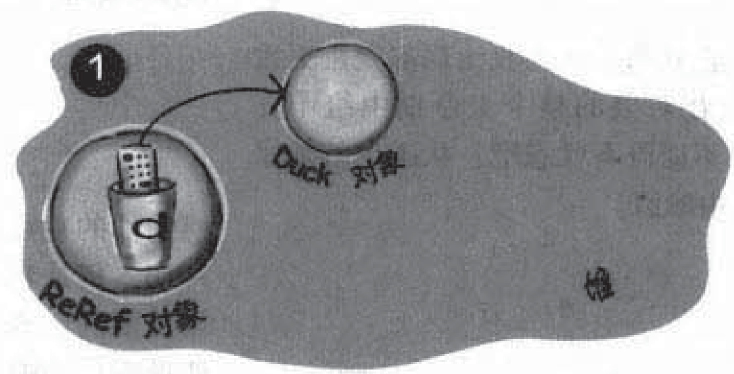
直接将引用设定为null



```
public class ReRef {

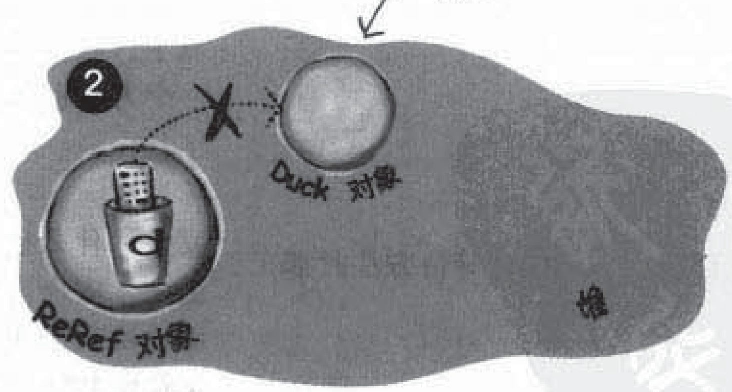
    Duck d = new Duck();

    public void go() {
      d = null;
    }
}
```



新的Duck会放在堆上，只要ReRef还活着，d就没事，除非……

它的唯一引用被设定为null



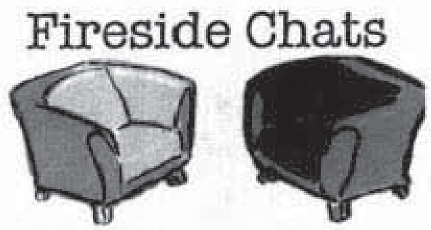
d被设定成null，就好像没有操作对象的遥控器

null的真相

当你把引用设为null时，你就等于是抹除遥控器的功能。换句话说，你会拿到一个没有电视的遥控器。null是代表"空"的字节组合（实际上是什么只有Java虚拟机才会知道）。

如果你真地按下这种遥控器上的按钮，什么事情也不会发生。但在Java上，你是不能对null引用按钮的。因为Java虚拟机会知道（这是运行期，不是编译时的错误）你期待喵喵叫，但是却没有Cat可以执行！

对null引用使用圆点运算符会在执行期遇到NullPointerException这样的错误。后面会有讨论异常的章节。



今晚的话题：

实例变量与局部变量的生死对谈

实例变量

我想应该先从我开始，因为我比局部变量更重要。通常我会在对象的整个生命期中给予支持。毕竟对象不能没有状态吧？状态也就是保持在实例变量中的值。

局部变量

感谢你的观点，也感谢你对对象状态值所做的一切。但是我不想让大家误会。局部变量是相当重要的。容我引用你的说法："毕竟对象不能没有行为吧？行为也就是保持在方法中的算法"。你也一定很清楚必须要有些局部变量才能让算法运作。

实例变量

误会可大了。我了解你在方法中的作用，只是你的命真的太短了。那也就是为何人们把你叫做"临时变量"的原因。

局部变量

留点口德吧，"临时变量"在我们这边是很轻蔑的说法。我们比较喜欢自称"局部变量"、"栈变量"或者"变量作用域"。

实例变量

抱歉，我知道了。

局部变量

算了。我们是不长命，但有时一个方法调用另一个方法也会让我们待在堆栈上很久。

实例变量

我倒不知道这回事，那你们在等待方法时都在做什么？

局部变量

发呆，什么也不做。但是我们所保存的值不会丢失，安全得很。要等到执行回到我们所处的堆栈块我们才会继续活动，不过这也代表我们又往生命终点迈进了一步。

实例变量

局部变量

我看过你同伴出殡的新闻报导，看起来蛮惨的。我的意思是当方法执行完成后，堆栈块就直接被清除，那种死法应该很痛吧。

你还说呢。这叫做被弹出好吗？为什么不来说说你呢？我生存在堆栈上，你老兄生存哪啊？

我跟对象一起生存在堆上，离西贡街与公众四方街口不远……嗯，其实是住在给我保存状态的对象里面才对。那附近地段很贵，物价也很高。

但你也不一定会很长寿吧？例如说你是猫对象身上的跳蚤实例变量所引用的跳蚤对象。假使今天这猫执行了洗澡的行为，使得跳蚤属性被设定为null，那会发生什么事？

对啦，如果我是个猫对象上的实例变量所引用的跳蚤对象，当该变量被设定成null的时候，我就只好等着被收进垃圾堆，我的地方也会让出来给别人用。不过有人跟我说过，这猫不可能会洗澡的。

你就这么相信了？如果猫对象只有一个引用呢？假使这个唯一的引用是保存在局部变量中呢？如果声明此局部变量的方法执行完毕，你还不是得像我们一样。老实跟你说，我已经认命了，现在能够活一天就活一天，有机会喝喝酒、吃吃RAM，我就尽量吃喝。

你不怕遇到警察吗？

我有一招可以逃过警察，你可以学学。如果你遇到警察的时候，就把……





我是垃圾收集器

假设批注部分实际上会是执行足够长时间的过程调用，下方右边有哪几行程序代码加到左边A位置会使得某一个额外的对象被认为是可以垃圾回收的？

```
public class GC {
  public static GC doStuff() {
    GC newGC = new GC();
    doStuff2(newGC);
    return newGC;
  }

  public static void main(String [] args) {
    GC gc1;
    GC gc2 = new GC();
    GC gc3 = new GC();
    GC gc4 = gc3;
    gc1 = doStuff();

    (A)

    // 调用更多的方法
  }

  public static void doStuff2(GC copyGC) {
    GC localGC
  }
}
```

1 `copyGC = null;`

2 `gc2 = null;`

3 `newGC = gc3;`

4 `gc1 = null;`

5 `newGC = null;`

6 `gc4 = null;`

7 `gc3 = gc2;`

8 `gc1 = gc4;`

9 `gc3 = null;`



最受欢迎金对象奖

下面的程序代码新建出数个对象。你的任务是要找出“最受欢迎”的对象，也就是被最多变量所引用的对象。还有，标记出所有对象以及对它的引用。我们已经先帮你标记出一个对象。

```
class Bees {
  Honey [] beeHA;
}

class Raccoon {
  Kit k;
  Honey rh;
}

class Kit {
  Honey kh;
}

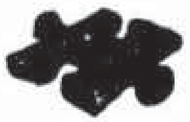
class Bear {
  Honey hunny;
}

public class Honey {
  public static void main(String [] args) {
    Honey honeyPot = new Honey();
    Honey [] ha = {honeyPot, honeyPot, honeyPot, honeyPot};
    Bees b1 = new Bees();
    b1.beeHA = ha;
    Bear [] ba = new Bear[5];
    for (int x=0; x < 5; x++) {
      ba[x] = new Bear();
      ba[x].hunny = honeyPot;
    }
    Kit k = new Kit();
    k.kh = honeyPot;
    Raccoon r = new Raccoon();

    r.rh = honeyPot;
    r.k = k;
    k = null;
  }  // main函数结束
}
```

新的Raccoon对象

引用它的变量r



“我们已经测了4次，主要模块的温度还是持续下降”，秋香不耐烦地说着，“上礼拜就已经安装新的温度感应器，散热器（radiator）的读数应该也没有问题，所以我们把焦点放在恒温器（retention bot）上”。华安叹了一口气，想起一开始的时候还以为纳米技术可以帮他们超前进度。现在离发射只剩下5个礼拜，卫星的生存维护装置还是没有办法通过测试。

华安传奇实录

“你用什么比例来模拟？”，华安问到。

“我知道你要问什么”，秋香回答，“我们也想到了。如果不符合规格，任务控制中心不会验收的。我们必须以2:1的比例执行v3版和v2版的radiator测试单元”，秋香继续说，“整体来说，retention与radiator的比例应该是4:3”。



华安追问：“能源消耗呢？”。秋香想了一下：“那是另外一问题，能源消耗速度比预期的快。我们有另外一组人正在想办法。但是这些无线技术很难将radiator的能源消耗与retention的消耗分离”。

秋香停了一下又继续说：“设计上的整体能源消耗率应该是3:2，也就是radiator的能源消耗会比较快”。

“好吧，既然这样”，华安说，“那我们再看一下仿真程序初始化的程序代码。这个问题一定得要很快地解决！”

```
import java.util.*;
class V2Radiator {
  V2Radiator(ArrayList list) {
    for(int x=0; x<5; x++) {
      list.add(new SimUnit("V2Radiator"));
    }
  }
}

class V3Radiator extends V2Radiator {
  V3Radiator(ArrayList lglist) {
    super(lglist);
    for(int g=0; g<10; g++) {
      lglist.add(new SimUnit("V3Radiator"));
    }
  }
}

class RetentionBot {
  RetentionBot(ArrayList rlist) {
    rlist.add(new SimUnit("Retention"));
  }
}
```

华安传奇实录（续）

```
public class TestLifeSupportSim {
  public static void main(String [] args) {
    ArrayList aList = new ArrayList();
    V2Radiator v2 = new V2Radiator(aList);
    V3Radiator v3 = new V3Radiator(aList);
    for(int z=0; z<20; z++) {
      RetentionBot ret = new RetentionBot(aList);
    }
  }
}

class SimUnit {
  String botType;
  SimUnit(String type) {
    botType = type;
  }
  int powerUse() {
    if ("Retention".equals(botType)) {
      return 2;
    } else {
      return 4;
    }
  }
}
```

把程序看了一遍之后，华安露出了得意的笑容。他说：“我想我知道那是怎么回事了。能源消耗这件事情其实也是有关的！”

到底华安看出哪里有问题？那要如何解决这个bug呢？

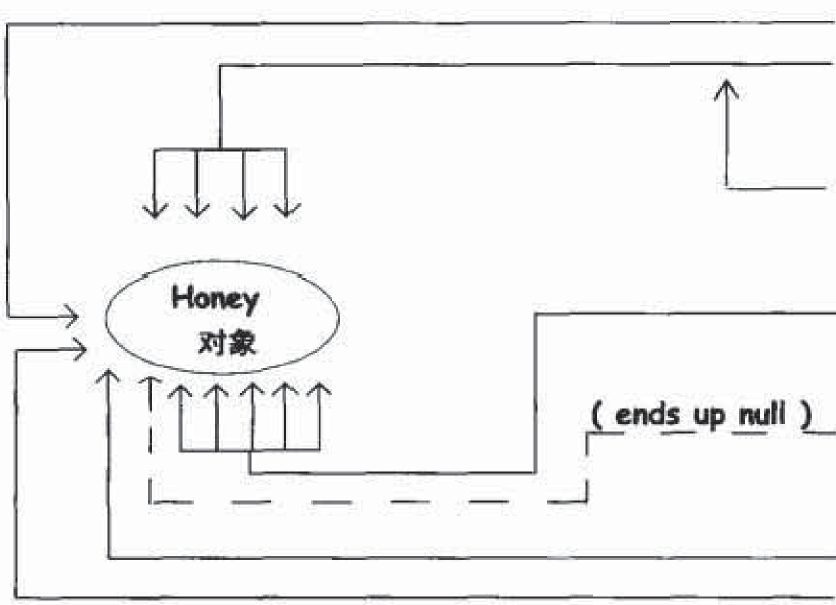


G.C.

1	`copyGC = null;`	**不行—— 这一行程序尝试要存取已经不在范围内的变量。**
2	`gc2 = null;`	**可以—— gc2是该对象唯一的引用。**
3	`newGC = gc3;`	**不行—— 也是超出范围。**
4	`gc1 = null;`	**可以—— gc1是唯一的引用。**
5	`newGC = null;`	**不行—— newGC已经超出范围。**
6	`gc4 = null;`	**不行—— 还有gc3引用该对象。**
7	`gc3 = gc2;`	**不行—— 还有gc4引用该对象。**
8	`gc1 = gc4;`	**可以—— 重新给对象引用赋值。**
9	`gc3 = null;`	**不行—— gc4还在引用该对象。**

最受欢迎金对象奖

要指出Honey这个对象是这个类中最受欢迎的对象应该不难。但要看出这些变量都指向同一个对象其实是要动点脑筋的。




```
public class Honey {
  public static void main(String [] args) {
    Honey honeyPot = new Honey();
    Honey [] ha = {honeyPot, honeyPot,
                   honeyPot, honeyPot};
    Bees b1 = new Bees();
    b1.beeHA = ha;
    Bear [] ba = new Bear[5];
    for (int x=0; x < 5; x++) {
      ba[x] = new Bear();
      ba[x].hunny = honeyPot;
    }
    Kit k = new Kit();
    k.kh = honeyPot;
    Raccoon r = new Raccoon();

    r.rh = honeyPot;
    r.k = k;
    k = null;
} }   // main函数结束
```

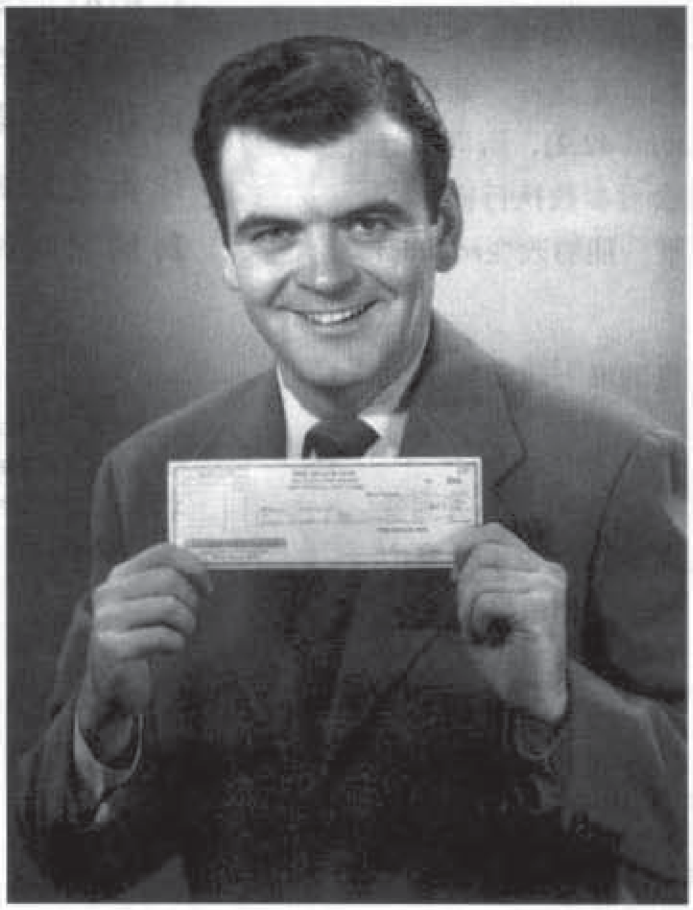
华安
传奇实录

华安发现V2Radiator类的构造函数会取用一个ArrayList参数。这代表每次V3Radiator的构造函数被调用时，它会在对V2Radiator的super()调用中传入一个ArrayList。这样会额外多出5个V2Radiator的SimUnit。如此一来，总体能源消耗会是120而不是秋香预期的100。

因为每个Bot都会创建出SimUnit，所以在SimUnit的构造函数中加上一栏输出就能够很快的发现问题的来源！

10 数字与静态

数字很重要



盘算一下吧。 除了primitive主数据类型的运算之外，数字还有其他的工作。你可能需要对数字计算绝对值、取整。或许需要以小数点后两位的打印格式，或者每隔三位数加上逗点以方便阅读。还有日期也是，你可能需要用某种特定的格式来打印日期，或者对日期作运算，例如说“今天起的两个礼拜后”。此外，字符串要如何转换成数字呢？幸好Java API中有很多与数字有关的方法能够很方便地使用。但这些方法多为静态的，因此我们会先从静态的变量和方法开始说起——这也包括了静态的final变量这种Java常数。

Math方法：最接近全局的方法

虽然在Java中没有东西是全局（global）的。但可以这么想：一种方法的行为不依靠实例变量值。例如Math这个类中的round()方法。它永远都执行相同的工作——取出浮点数（方法的参数）的最接近整数值。永远都是这样。如果你有10000个Math的实例，并且都执行round(42.2)，所得的值永远都会是42。换句话说，这个方法会对参数执行操作，但这操作不受实例变量状态的影响。唯一能够改变round()行为的只有所传入的参数。

你不觉得为了要执行round()而得在宝贵的堆上建立Math的实例是很浪费的事吗？

像round()、abs()、max()等数学运算方法其实不需要实例变量值。事实上也不会有Math的实例变量。因此也不会有空间被它的实例所占用。你猜怎么样？你不需创建Math的实例，实际上你也无法创建。

在Math这个类中的所有方法都不需要实例变量值。因为这些方法都是静态的，所以你无需Math的实例。你会用到的只有它的类本身。

```
int x = Math.round(42.2);
int y = Math.min(56,12);
int z = Math.abs(-343);
```

这些方法无需实例变量，因此也不需要特定对象来判别行为。

如果你硬要创建Math的实例：

```
Math mathObject = new Math();
```

会得到下面这样的错误信息：

```
File Edit Window Help IwasToldThereWouldBeNoMath
%javac TestMath

TestMath.java:3: Math() has private
access in java.lang.Math

    Math mathObject = new Math();
                      ^

1 error
```

错误信息指出Math的构造函数被标记为私有的！这代表你不能新建Math的对象

非静态方法与静态方法的差别

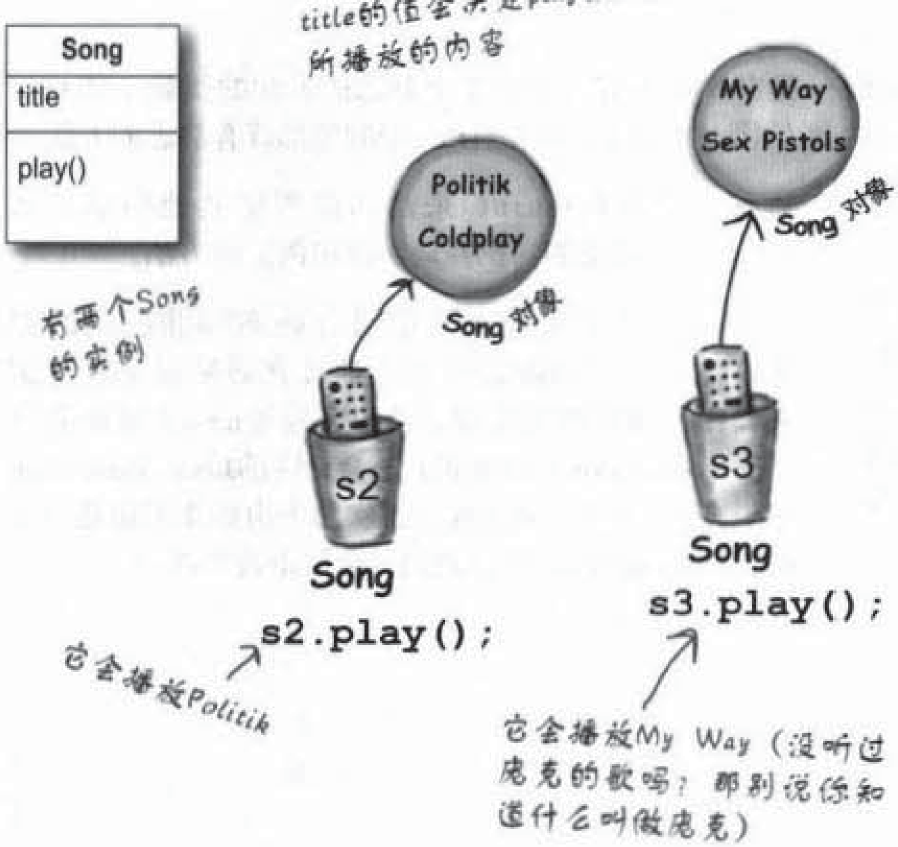
Java是面向对象的，但若处于某种特殊的情况下，通常是实用方法，则不需要类的实例。static这个关键词可以标记出不需类实例的方法。一个静态的方法代表说“一种不依靠实例变量也就不需要对象的行为”。

非静态方法

```
public class Song {
   String title;
   public Song(String t) {
     title = t;
   }
   public void play() {
      SoundPlayer player = new SoundPlayer();
      player.playSound(title);
   }
```

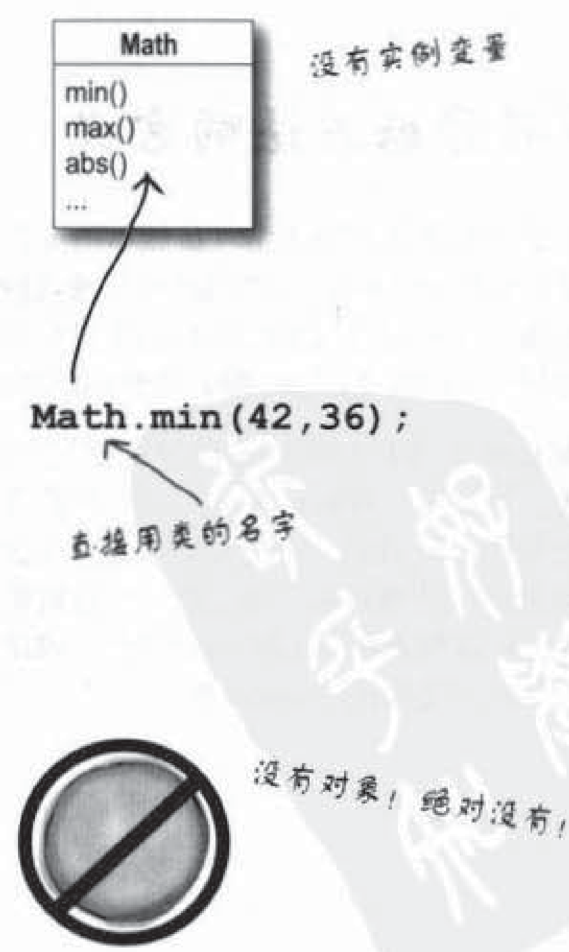
实例变量的值会影响到play()方法的行为

title的值会决定play()方法所播放的内容

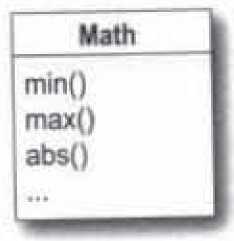


静态方法

```
public static int min(int a, int b){
    // 返回a与b中较小的值
}
```



以类的名称调用静态的方法




```
Math.min(88,86);
```

以引用变量的名称调用非静态的方法




```
Song t2 = new Song();
t2.play();
```

带有静态方法的含义

带有静态的方法的类通常（虽然不一定是这样）不打算要被初始化。在第8章我以已经讨论过抽象类，以及如何用abstract这个修饰字来标记类以让它不能被创建出实例。换句话说，抽象的类是不能被初始化的。

但你也可以用私有的构造函数来限制非抽象类被初始化。要记得，被标记为private的方法代表只能被同一类的程序所调用。构造函数也是同样意思，这也是Math如何防止被初始化的方法。它让构造函数标记为私有，所以你无法创建Math的实例。编译器会知道你不能存取这些私有的构造函数。

这并不是说有一个或多个静态的方法的类就不能被初始化。事实上，只要有main()的类都算有静态的方法！

通常你会写出main()来启动或测试其他的类。从main()中创建类的实例并调用新实例上的方法。

因此你可以任意地在类中组合静态与非静态的方法，然而任何非静态的方法都代表必须以某种实例来操作。取得新对象的方法只有通过new或者序列化（deserialization）以及我们不会讨论的Java Reflection API。除此之外，别无他法。实际上由谁来新建是一个很有意思的问题，稍后我们就会讨论这个部分。

静态的方法不能调用非静态的变量

静态的方法是在无关特定类的实例情况下执行的。如同你在上一页所看到的，甚至也不会有该类的实例出现。因为静态的方法是通过类的名称来调用，所以静态的方法无法引用到该类的任何实例变量。在此情况下，静态的方法也不会知道可以使用哪个实例变量值。

如果你尝试在静态的方法内使用实例变量，编译器会认为："我不知道你说的是哪个实例的变量！"

静态的方法是不知道堆上有哪些实例的。

如果你要编译下面这段程序代码：

```
public class Duck {

   private int size;

   public static void main (String[] args) {
      System.out.println("Size of duck is " + size);
   }

   public void setSize(int s) {
      size = s;
   }
   public int getSize() {
       return size;
   }
}
```

哪一个Duck？

此时我们根本无从得知堆上是否有Duck



你会得到这样的错误：

```
File Edit Window Help Quack
% javac Duck.java
Duck.java:6: non-static variable
size cannot be referenced from a
static context
      System.out.println("Size
of duck is " + size);
                ^
```

静态的方法也不能调用非静态的方法

非静态的方法做什么工作？它们通常是以实例变量的状态来影响该方法的行为。getName()方法会返回name变量的值。谁的名字？当然是被调用对象的。

这一段无法编译：

```
public class Duck {

   private int size;

   public static void main (String[] args) {
      System.out.println("Size is " + getSize());
   }

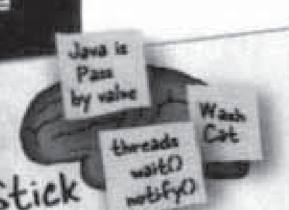
   public void setSize(int s) {
      size = s;
   }
   public int getSize() {
       return size;
   }
}
```

调用getSize()会需要用到size实例变量

老问题，到底是谁的size?

```
File Edit Window Help Jack-in
% javac Duck.java
Duck.java:6: non-static method
getSize() cannot be referenced
from a static context
      System.out.println("Size
of duck is " + getSize());
                ^
```

Make it Stick



玫瑰是红的，

浮尘是不落地的。

静态方法无法看到实例变量的状态。

there are no Dumb Questions

问： 如果从静态方法调用非静态方法，但此非静态方法没有用到实例变量，这样会通过编译吗？

答： 不会。编译器可以知道你有没有使用实例变量，你也知道。但如果现在可以通过，后来却把非静态变量改成会使用实例变量呢？又如果子类去覆盖这个方法成有用到实例变量的版本呢？

问： 我发誓曾经看过以引用变量代替类名称调用静态方法的程序代码，这样对吗？

答： 你确实可以这样做，但合法的事情并不一定都是好事。虽然可以使用类的实例来调用，但这样会产生容易误解的程序代码：

```
Duck d = new Duck( );
String[] s = { };
d.main(s);
```

这段程序代码是合法的，但编译器还是会解析出原来的类。使用d来调用main()并不代表main会知道是哪个对象引用所做的调用。如此调用的方法也还是静态的！

静态变量：它的值对所有的实例来说都相同

假设你要在执行过程中计算有多少Duck的实例已经被建立出来。你要怎么做？或许可以在构造函数中递增某个实例变量的值？

```
class Duck {
   int duckCount = 0;
    public Duck() {
       duckCount++;   这会在创建Duck对象时
    }                 执行递增
}
```

不行，因为duckCount是个实例变量，所以这样做不会成功。每个Duck在初始化的时候duckCount的值都是0。你也许可以调用别的类来计算，不过这样又不太优雅。你需要的是只会有一份拷贝的变量，且所有实例都会用到该拷贝。

这就是静态变量的功用：被同类的所有实例共享的变量。

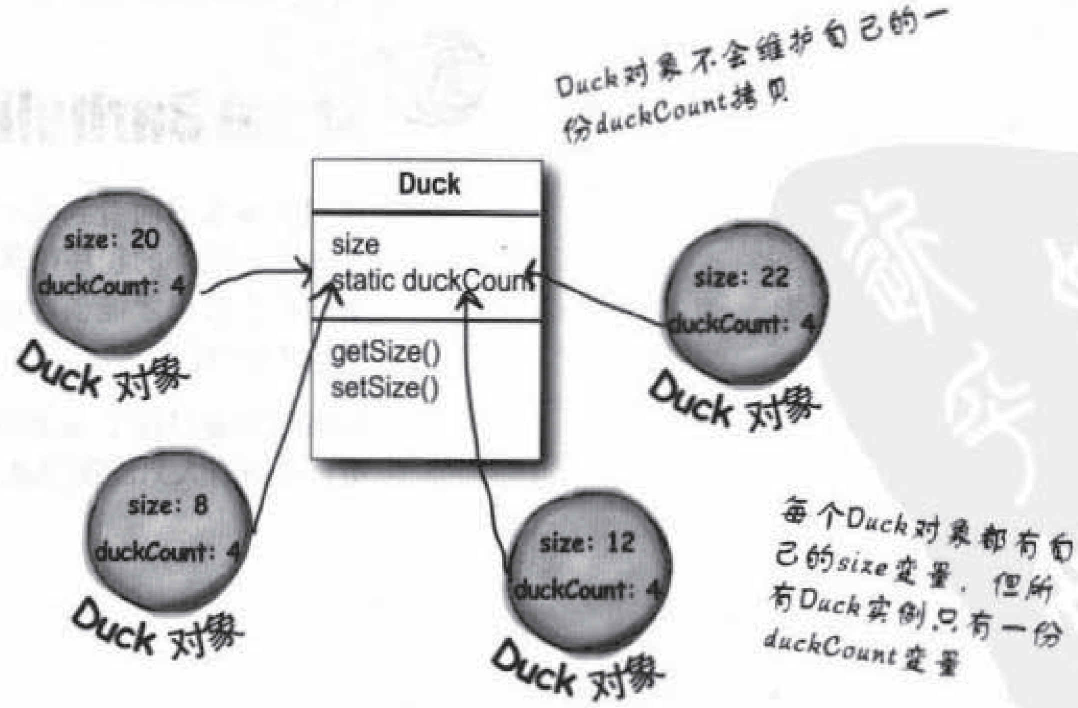
此静态的duckCount变量只会在类第一次载入的时候被初始化。

```
public class Duck {

   private int size;
   private static int duckCount = 0;

   public Duck() {   ← 每当构造函数执行的时候，此变量的值就会递增
      duckCount++;
   }

   public void setSize(int s) {
      size = s;
   }
   public int getSize() {
      return size;
   }
}
```





静态变量是共享的。

同一类所有的实例共享一份静态变量。

实例变量： 每个实例一个。
静态变量： 每个类一个。



Brain Barbell

我们在本章前面看到私有的构造函数代表这个类不能被本身以外的程序给实例化。也就是说，只有此类的程序代码才能创建出新的实例（到底是鸡生蛋还是蛋生鸡？）。

如果你想要以这个方法编写出只能创建一个实例的类，且所有人只能运用这一份实例，要该怎么做？

静态变量的起始动作

静态变量是在类被加载时初始化的。类会被加载是因为Java虚拟机认为它该被加载了。通常，Java虚拟机会加载某个类是因为第一次有人尝试要创建该类的新实例，或是使用该类的静态方法或变量。程序员其实也可以选择强制Java虚拟机去加载某个类，但你不太需要这么做。大部分的情况下还是让Java虚拟机来决定会比较好。

静态变量会在该类的任何静态方法执行之前就初始化。

静态项目的初始化有两项保证：

静态变量会在该类的任何对象创建之前就完成初始化。

静态变量会在该类的任何静态方法执行之前就初始化。

```
class Player {
    static int playerCount = 0;
    private String name;
    public Player(String n) {
        name = n;
        playerCount++;
    }
}
```

playerCount会在载入类的时候被初始化为0。

像是long或short等primitive主数据类型整数的默认值是0，primitive主数据类型的浮点数默认值是0.0，boolean是false，对象引用是null

```
public class PlayerTestDrive {
    public static void main(String[] args) {
        System.out.println(Player.playerCount);
        Player one = new Player("Tiger Woods");
        System.out.println(Player.playerCount);
    }
}
```

静态变量也是通过类的名称来存取

```
File Edit Window Help What?
% java PlayerTestDrive
0
1
```

0 ← 在实例创建之前

1 ← 对象创建之后

如果你没有给静态变量赋初值，它就会被设定默认值。int会被设定为0。静态变量的默认值会是该变量类型的默认值，就像实例变量所被赋予的默认值一样。

静态的final变量是常数

一个被标记为final的变量代表它一旦被初始化之后就不会改动。也就是说类加载之后静态final变量就一直会维持原值。以Mathp.PI为例：

public static final double PI = 3.141592653589793;

此变量被标记为public，因此可供各方读取。

此变量被标记为static，所以你不需要Math的实例。

此变量被标记为final，因为圆周率是不变的。

此外没有别的方法可以识别变量为不变的常数（constant），但有命名惯例（naming convention）可以帮助你认出来。

常数变量的名称应该要都是大写字母！

静态初始化程序（static initializer）是一段在加载类时会执行的程序代码，它会在其他程序可以使用该类之前就执行，所以很适合放静态 final 变量的起始程序。

```
class Foo {
    final static int x;
    static {
      x = 42;
    }
}
```

静态 final 变量的初始化：

① 声明的时候：

```
public class Foo {
   public static final int FOO_X = 25;
}
```

注意这个命名惯例——应该都是大写的，并以下划线字符分隔

或

② 在静态初始化程序中：

```
public class Bar {
   public static final double BAR_SIGN;

   static {
       BAR_SIGN = (double) Math.random();
   }
}
```

这段程序会在类被加载时执行

如果你没有以这两种方式之一来给值的话：

```
public class Bar {
   public static final double BAR_SIGN;
}
```

没有初始化！

编译器会发现这个问题：

```
File Edit Window Help Jack-in
% javac Bar.java
Bar.java:1: variable BAR_SIGN
might not have been initialized
1 error
```

final不只用在静态变量上……

你也可以用final关键字来修饰非静态的变量，这包括了实例变量、局部变量甚或是方法的参数。不管哪一种，这都代表它的值不能变动。但你也可以用final来防止方法的覆盖或创建子类。

非静态final变量

```
class Foof {
    final int size = 3;  ← size将无法改变
    final int whuffie;

    Foof() {
        whuffie = 42;  ← whuffie不能改变
    }

    void doStuff(final int x) {
        // 不能改变x
    }

    void doMore() {
        final int z = 7;
        // 不能改变z
    }
}
```

final 的 method

```
class Poof {
    final void calcWhuffie() {
        // 绝对不能被覆盖过
    }
}
```

final 的 class

```
final class MyMostPerfectClass {
    // 不能被继承过
}
```

final的变量代表你不能改变它的值。

final的method代表你不能覆盖掉该method。

final的类代表你不能继承该类（也就是创建它的子类）。



there are no Dumb Questions

问： 静态的方法不能存取非静态的变量，但非静态的方法可以读取静态的变量吗？

答： 当然可以。非静态方法不可以调用该类静态的方法或静态的变量。

问： 为何需要将类标记为final？这不会破坏面向对象的目的吗？

答： 会也不会。将类标记为final的主要目的是为了安全。例如String这个类，假使有人继承过，弄了一个行为很不一致的版本，就会对预期操作String的程序产生很多问题。

问： 如果类已经是final的，再标记final的方法是不是很多余？

答： 不只是多余，而且多了很多。如果一个类不能被子类化，则它的方法根本就无法被覆盖。如果只是想要限制部分的方法不能被覆盖过，那就单独地标记它们为final的就行。

要点

- 静态的方法应该用类的名称来调用，而不是用对象引用变量。
- 静态的方法可以直接调用而不需要堆上的实例。
- 静态的方法是一个非常实用的方法，它不需特别的实例变量值。
- 静态的方法不能存取非静态的方法。
- 如果类只有静态的方法，你可以将构造函数标记为private的以避免被初始化。
- 静态变量为该变量所属类的成员所共享。静态变量只会有一份，而不是每个实例都有自己的一份。
- 静态方法可以存取静态变量。
- 在Java中的常量是把变量同时标记为static和final的。
- final的静态变量值必须在声明或静态初始化程序中赋值：

```
static {
    DOG_CODE = 420;
}
```

- 常量的命名惯例是全部使用大写字母。
- final 值一旦被赋值就不能更改。
- final 的方法不能被覆盖。
- final 的类不能被继承。

Sharpen your pencil

谁是合法的？

就你所学到的static和final知识来看，下面哪些程序可以通过编译？




```
public class Foo {
   static int x;

   public void go() {
      System.out.println(x);
   }
}
```

```
public class Foo2 {
  int x;

  public static void go() {
      System.out.println(x);
  }
}
```

```
public class Foo3 {
  final int x;

  public void go() {
      System.out.println(x);
  }
}
```

```
public class Foo4 {
  static final int x = 12;

  public void go() {
      System.out.println(x);
  }
}
```

```
public class Foo5 {
  static final int x = 12;

  public void go(final int x) {
      System.out.println(x);
  }
}
```

```
public class Foo6 {
  int x = 12;

  public static void go(final int x) {
      System.out.println(x);
  }
}
```

Math 的方法

现在我们已经知道static的方法是如何工作的，接着让我们来看一下Math的一些方法。这不是全部，API文件还有像是sqrt()、tan()、celi()等的说明。

Math.random()

返回介于0.0～1.0之间的双精度浮点数。

```
double r1 = Math.random();
int r2 = (int) (Math.random() * 5);
```

Math.abs()

返回双精度浮点数类型参数的绝对值。这个方法有覆盖的版本，因此传入整型会返回整型，传入双精度浮点数会返回双精度浮点数。

```
int x = Math.abs(-240);  // 返回240
double d = Math.abs(240.45);  // 返回240.45
```

Math.round()

根据参数是浮点型或双精度浮点数返回四舍五入之后的整型或长整型值。

```
int x = Math.round(-24.8f);  // 返回-25
int y = Math.round(24.45f);  // 返回24
```

↑ 文字直接表示的浮点数都会被当作双精度浮点数，除非后面有加上f

Math.min()

返回两参数中较小的那一个。这有int、long、float或double的覆盖版本。

```
int x = Math.min(24,240);  // 返回24
double y = Math.min(90876.5, 90876.49);  // 返回90876.49
```

Math.max()

返回两参数中较大的那一个。这有int、long、float或double的重载版本。

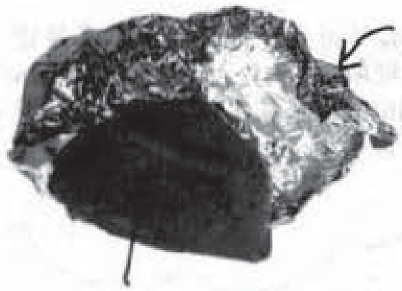
```
int x = Math.max(24,240);  // 返回240
double y = Math.max(90876.5, 90876.49);  // 返回90876.5
```

primitive主数据类型的包装

有时你会想要把primitive主数据类型当作对象来处理。例如在5.0之前的Java版本上，你无法直接把primitive主数据类型放进ArrayList或HashMap中：

```
int x = 32;
ArrayList list = new ArrayList();
list.add(x);
```

除非是用Java 5.0或以上的版本，否则这个命令不会成功



每一个primitive主数据类型都有个包装用的类，且因为这些包装类都在java.lang这个包中，所以你无需去import它们。每个包装类都很好辨别，因为它的名称是照着所包装的类型所设定的，只是将第一个字母改为大写以符合命名惯例。

当你需要以对象方式来处理primitive主数据类型时，就把它包装起来。Java 5.0之前的版本必须要这么做。

还有，为了某些没有人知道的理由，API的设计者决定让名称不是完全地符合primitive主数据类型的名称：

Boolean
Character
Byte
Short
Integer
Long
Float
Double

注意！这个名称与primitive主数据类型不同，是完整拼出来的



包装值

```
int i = 288;
Integer iWrap = new Integer(i);
```

传入primitive主数据类型给包装类的构造函数

解开包装

```
int unWrapped = iWrap.intValue();
```

所有的包装类都有类似这样的方法

*上面的图听说是个叉烧包



在Java 5.0以前你得这样做……

没错。在Java 5.0之前的版本上，primitive主数据类型就是原始类型，而对象引用就是对象引用，两者绝无交换使用的方法。要交互使用就得靠程序员进行包装与拆开包装的动作。没有办法能够以primitive主数据类型来直接传入期待对象引用的方法，也没有办法能够把回传对象参考的method值赋值给primitive主数据变量。Interger与int两者间毫无关系可言，只是Integer带有一个int类型的实例变量（以保存Integer所包装的primitive）。你得想办法自己进行这一类转换的工作。

primitive int的ArrayList

无autoboxing

```
public void doNumsOldWay() {

    ArrayList listOfNumbers = new ArrayList();

    listOfNumbers.add(new Integer(3));

    Integer one = (Integer) listOfNumbers.get(0);

    int intOne = one.intValue();
}
```

创建出ArrayList对象

不能直接加入primitive的3，得先转换成Integer

返回Object类型，但你可以将Object转换成Integer

最后再取出primitive

autoboxing：不必把primitive主数据类型与对象分得那么清楚

从5.0版开始加入的autoboxing功能能够自动地将primitive主数据类型转换成包装过的对象！

让我们看一下创建int的ArrayList时会发生什么事。

primitive int的ArrayList

有autoboxing

```
public void doNumsNewWay() {

    ArrayList<Integer> listOfNumbers = new ArrayList<Integer>();

    listOfNumbers.add(3);

    int num = listOfNumbers.get(0);
}
```

创建Integer类型的ArrayList

直接加！

虽然ArrayList没有add(int)这样的方法，但编译器会自动帮你包装。

编译器也会自动地解开Integer对象的包装，因此可以直接赋值给int。

问： 为什么不直接声明 ArrayList<int>？

答： generic类型的规则是你只能指定类或接口类型。因此ArrayList<int>将无法通过编译。但你可以直接把该包装所对应的primitive主数据类型放进ArrayList中，例如说boolean类型的放入ArrayList<Boolean>中chars类型的放入ArrayList<Character>中。

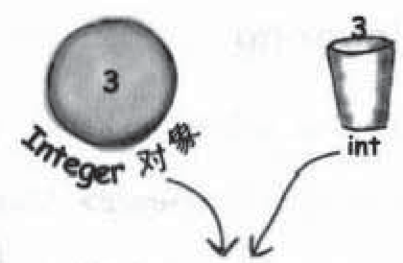
到处都用得到autoboxing

autoboxing不只是包装与解开primitive主数据类型给collection用而已，它还可以让你在各种地方交换地运用primitive主数据类型与它的包装类型。想想看有哪些地方会用到。

autoboxing乐趣多

方法的参数

如果参数是某种包装类型，你可以传入相对应的primitive主数据类型，反之亦然。

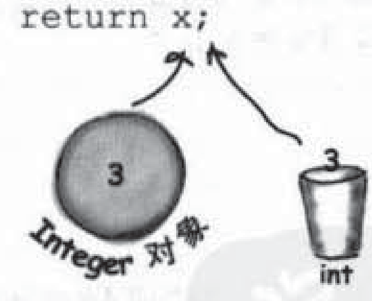



```
void takeNumber(Integer i) { }
```

返回值

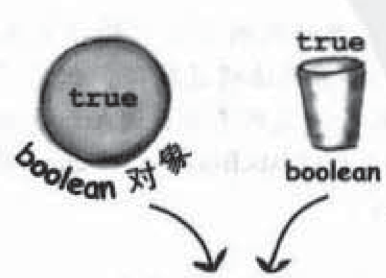
如果method声明为返回某种primitive主数据类型，你也可以返回兼容的primitive主数据类型或该primitive主数据类型的包装类型。

```
int giveNumber() {
  return x;
}
```



boolean 表达式

任何预期boolean值的位置都可以用求出boolean的表达式来代替，例如说4>2或是Boolean包装类型的引用。




```
if (bool) {
   System.out.println("true");
}
```

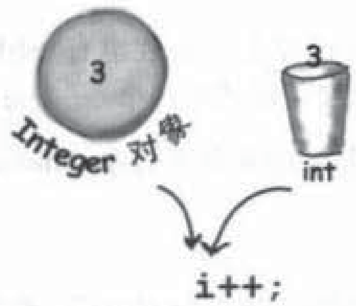
数值运算

这或许是最诡异的。你可以在使用primitive主数据类型作为运算子的操作中以包装类型来替换。这代表你可以对Integer的对象作递增运算!

Integer i = new Integer(42);
i++;

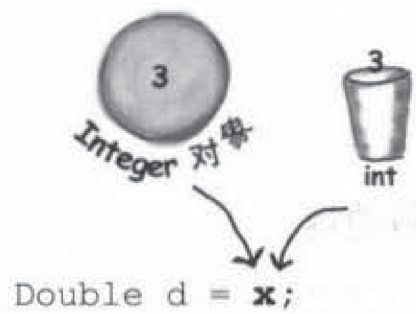
还可以这么做:

Integer j = new Integer(5);
Integer k = j + 3;



赋值

你可以将包装类型或primitive主数据类型赋给声明成相对应的包装或primitive主数据类型。



Sharpen your pencil

右边的程序代码能否通过编译?可以执行吗?如果可以,会有什么结果?

慢慢来,你有足够的时间去想,这会引起我们没讨论过的autoboxing问题。

你得要编译才会有答案,所以我们是在逼你上机操作。来吧!

```
public class TestBox {

   Integer i;
   int j;

   public static void main (String[] args) {
      TestBox t = new TestBox();
      t.go();
   }

   public void go() {
      j=i;
      System.out.println(j);
      System.out.println(i);
   }
}
```

等一下！还有呢！包装也有静态的实用性方法！

除了一般类的操作外，包装也有一组实用的静态方法。我们在之前已经用过其中一个——Integer.parseInt()。

这个方法取用String并返回给你primitive主数据类型值。

将String转换成primitive主数据类型值是很容易的：

将"2"解析成2

```
String s = "2";
int x = Integer.parseInt(s);
double d = Double.parseDouble("420.24");

boolean b = new Boolean("true").booleanValue();
```

你可能会以为有Boolean.parseBoolean()吧？其实没有。但是Boolean的构造函数可以取用String来创建对象

但若你这么做的话：

```
String t = "two";
int y = Integer.parseInt(t);
```

啊！可以通过编译，但执行时就会出状况

就会在运行期间遇到异常：

```
File Edit Window Help Clue
% java Wrappers
Exception in thread "main"
java.lang.NumberFormatException: two
at java.lang.Integer.parseInt(Integer.java:409)
at java.lang.Integer.parseInt(Integer.java:458)
at Wrappers.main(Wrappers.java:9)
```

解析String的方法或构造函数会抛出NumberFormatException异常。这是运行期间的异常，你应该会处理这种异常。

（下一章会讨论异常）

反过来将primitive主数据类型值转换成String

有好几种方法可以将数值转换成String。最简单的方法是将数字接上现有的String。

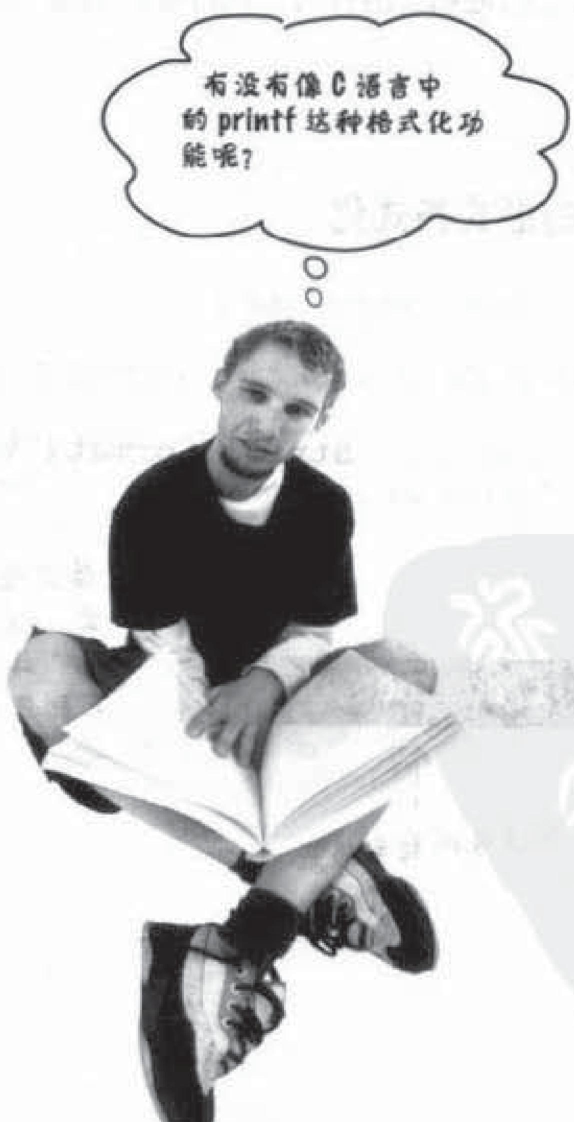
```
double d = 42.5;
String doubleString = "" + d;

double d = 42.5;
String doubleString = Double.toString(d);
```

"+"这个操作数是Java中唯一有重载过的运算符

Double这个类的静态方法





数字的格式化

在Java中，数字与日期的格式化功能并没有结合在输出/输入功能上。通常对用户显示数字是通过GUI来进行的。你会把String放在可滚动的文字区域块或表格中。如果格式化功能只有绑在文字模式输出的命令上，那就没有办法把字符串以比较漂亮的格式输出到GUI上。在Java 5.0之前的格式化功能是通过java.text这个包来处理，但本书已经不屑去提它了。

从Java 5.0起，更多更好更有扩展性的功能是通过java.util中的Formatter这个类来提供的。但你无需自己创建与调用这个class上的方法，因为Java 5.0已经把便利性的功能加到部分的输出/输入类与String上。因此只要调用静态的String.format()并传入值与格式设定就好。

当然，你还是得知道如何提供格式设定，本章会有一些基本的说明，我们会从基本的范例开始，并观察它们是如何运行的（在讨论输出/输入的章节中还会再看过一次）。

将数字以带逗号的形式格式化

```
public class TestFormats {

   public static void main (String[] args) {

      String s = String.format("%, d", 1000000000);
      System.out.println(s);
   }
}
```

要格式化的数字

格式设定，用来指示应该用哪种形式来输出，这里的逗号是表示数字要以逗号来分开，并不是说这里有%与d两项参数，千万别再错了！

```
1,000,000,000
```

有逗号的数字格式

解析格式化结构……

基本上来说，格式化由两个主要部分组成（不只是这样，但我们先从简单开始）：

- **格式指令**

  描述要输出的特殊格式。

- **要格式化的值**

  不是所有东西都能被格式化。例如，如果你的格式指令适用于浮点数，则你就不能传入Dog或看起来很像浮点数的String。

如果你本来就已经熟悉C/C++的printf()，那这几页说的全部都是废话！

以这种格式……　　来表现这个值

```
format("%, d", 100000000);
```

用这个格式将这个参数格式化

这个指令代表什么？

将此方法的第二个参数以第一个参数所表示带有逗号的整数（decimal）方式表示。

它会怎么做？

下一页会对“%, d”作更详细的说明，我们在这里先简略地说明一下。在格式化指令中的%代表一项变量，此变量就是跟在格式化指令后面的参数，其余的字符各有所代表的意义。

%符号代表把参数放在这里

format()方法的第一个参数被称为“格式化串”，它可以带有实际上就是要这么输出而不用转译的字符。当你看到%符号时，要把它想做是会被方法其余参数替换掉的位置。

这会原汁原味地输出

参数，也就是后面那个数字

这也会原汁原味地输出

要被处理的值

```
format("I have %.2f bugs to fix.", 476578.09876);
```

输出

```
I have 476578.10 bugs to fix.
```

注意：数值的精确度有损失！你看出来.2f的意义吗？

此例的%符号是第二个参数会放置的位置，“.2f”代表该参数要使用的格式，其他的字都会以原来的方式输出。

加上逗号

```
format("I have %,.2f bugs to fix.", 476578.09876);
```

```
I have 476,578.10 bugs to fix.
```

%符号后面的指令被加上逗号之后，输出也有了变化



格式化语句有自己的一套语法

很明显，%符号后面不可以随便填上任意的字符。%的语法有非常特殊的规则，是用来描述此处所用的格式。

你已经看过两个例子：

%,d：这代表以十进制整数带有逗号的方式来表示。

%.2f：这代表以小数点后两位的方式来格式化此浮点数。

%,.2f：代表整数部分以有逗号的形式表示，小数部分以两位来格式化。

所以问题在于你怎么知道什么字符代表什么意义，以及这些指令字符的使用顺序和时机。

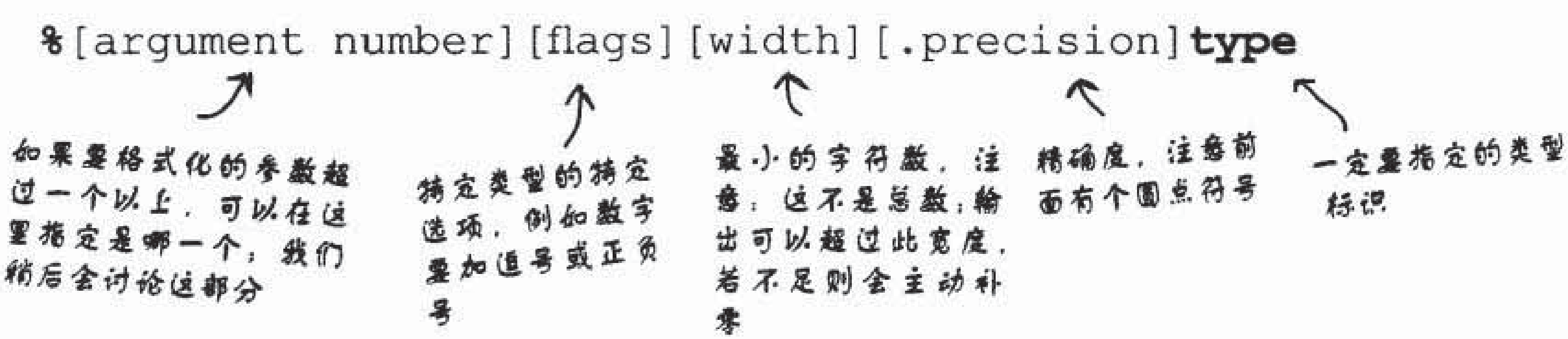
想想看下面这个语句会做出什么样的输出？答案在下一页：

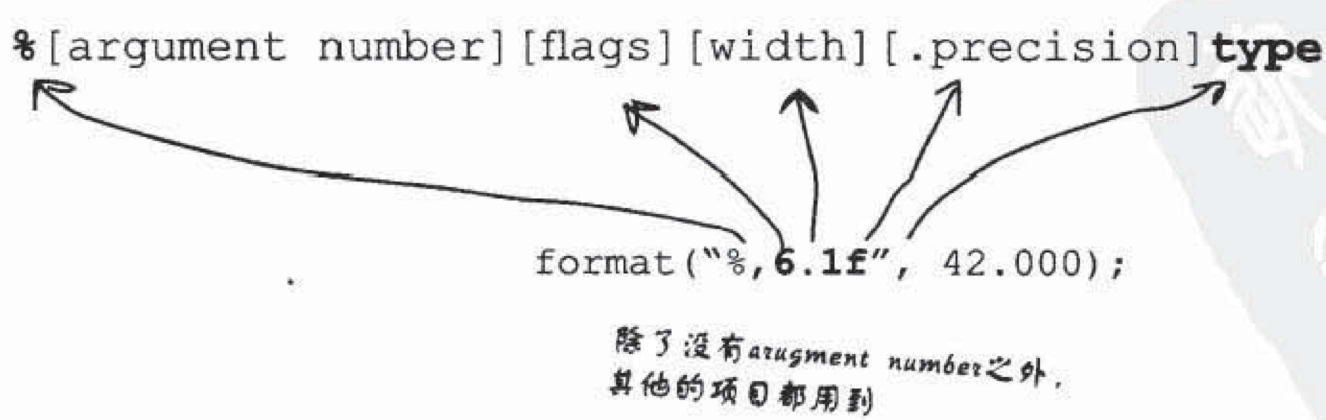
```
String.format("I  have %.2f, bugs to fix.", 476578.09876);
```

“格式化说明”的格式

跟在百分号后面包括类型指示（像是d或f）的每个东西都是格式化指令。除非遇到新的百分号，在类型指示之后的一组字符，格式化程序会假设都是直接输出的字符串。这可能吗？要被格式化的参数可以超过一个以上吗？先别管这个，稍后再讨论。现在先来看格式化说明的语法——跟在%后面的那些指令。

格式化说明最多会有5个部分（不包括%符号）。下面的[ ]符号里面都是选择性的项目，因此只有%与type是必要的。格式化说明的顺序是有规定的，必须要以这个顺序来指定。





唯一的必填项目是类型

虽然类型是唯一必填的项目，但若指定别的项目，则类型必须是最后一项！类型修饰符有十几种（这不包括日期和时间的，它们有自己的一组），但大部分时间你会使用到%d或%f。且通常你会对%f加上精确度指示来设定所需要的小数长度。

The TYPE is mandatory, everything else is optional.

在格式化指令中一定要给的类型，如果还要指定其他项目的话，要把类型摆在最后。

%d ***decimal***

```
format("%d", 42);
```

如果给的是42.45就不行

```
42
```

参数必须能够与 int 相容。

%f ***floating point***

```
format("%.3f", 42.000000);
```

.3配上f会强制输出3位的小数

```
42.000
```

参数必须是浮点数类型。

%x ***hexadecimal***

```
format("%x", 42);
```

```
2a
```

参数必须是byte、short、int、long、BigInteger。

%c ***character***

```
format("%c", 42);
```

ASCII的42代表"*"号

```
*
```

参数同上，但不包括BigInteger。

超过一项以上的参数时呢？

如果你要输出像下面这样的字符串：

"The rank is 20,456,654 out of 100,567,890.24."

但这两个数字来自于不同的变量，该怎么做？把新的参数加到后面，因此你会以3个参数来调用format()而不是两个。并且在第一个参数中，也就是格式化串中，会有两个不同的格式化设定，也就是两个%开头的字符组合。第二个参数会应用在第一个%号上面，第三个参数会用在第二组%组合上。也就是说参数会依照顺序应用在%上面。

```
int one = 20456654;
double two = 100567890.248907;
String s = String.format("The rank is %,d out of %,.2f", one, two);
```

```
The rank is 20,456,654 out of 100,567,890.25
```

一项以上的参数会依序对应到格式化设定

两项都有加逗号，且后者的小数被限制在两位

当我们讨论到日期的格式化时，你就会看到以同一个参数应用在不同的格式化设定上。这可能有点难以理解，除非你直接看到日期格式化的例子。接下来我们就会讨论如何明确地指定哪个格式化设定要用在哪个参数上。

问： **有些地方真的怪怪的，到底可以传多少个参数进去？我是说format()到底有多少个重载的版本？如果有10个参数要用在格式化上面会怎样？**

答： 问得好！没错，这里有些不一样的东西，且实际上是没有一大堆的重载版format()来取用不同数目排列组合的参数。为了要应付格式化的API，Java语言需要一种新的功能——称为可变参数列表（varable argument list，简称为vararg）。这个部分在附录中有说明，通常你很少会用到这样的功能。

都与数字有关，那日期呢？

如果你要输出这样的字符串：Sunday, Nov 28 2004

看起来好像没什么特别。但是如果说是要把Date类型的变量日期用这样的格式输出呢？Date类型是Java上表示时间用的，而现在你得处理这个类型。

数值与日期时间格式化的主要差别在于日期格式的类型是用“t”开头的两个字符来表示，下面有几个范例：

完整的日期与时间：%tc

```
String.format("%tc", new Date());
```

```
Sun Nov 28 14:52:41 MST 2004
```

只有时间：%tr

```
String.format("%tr", new Date());
```

```
03:01:47 PM
```

周、月、日：%tA %tB %td

因为没有刚好符合我们要求的输出格式，所以得组合3种形式来产生出所需要的格式：

```
Date today = new Date();
String.format("%tA, %tB %td",today,today,today);
```

这样就得把Data对象传进去3次

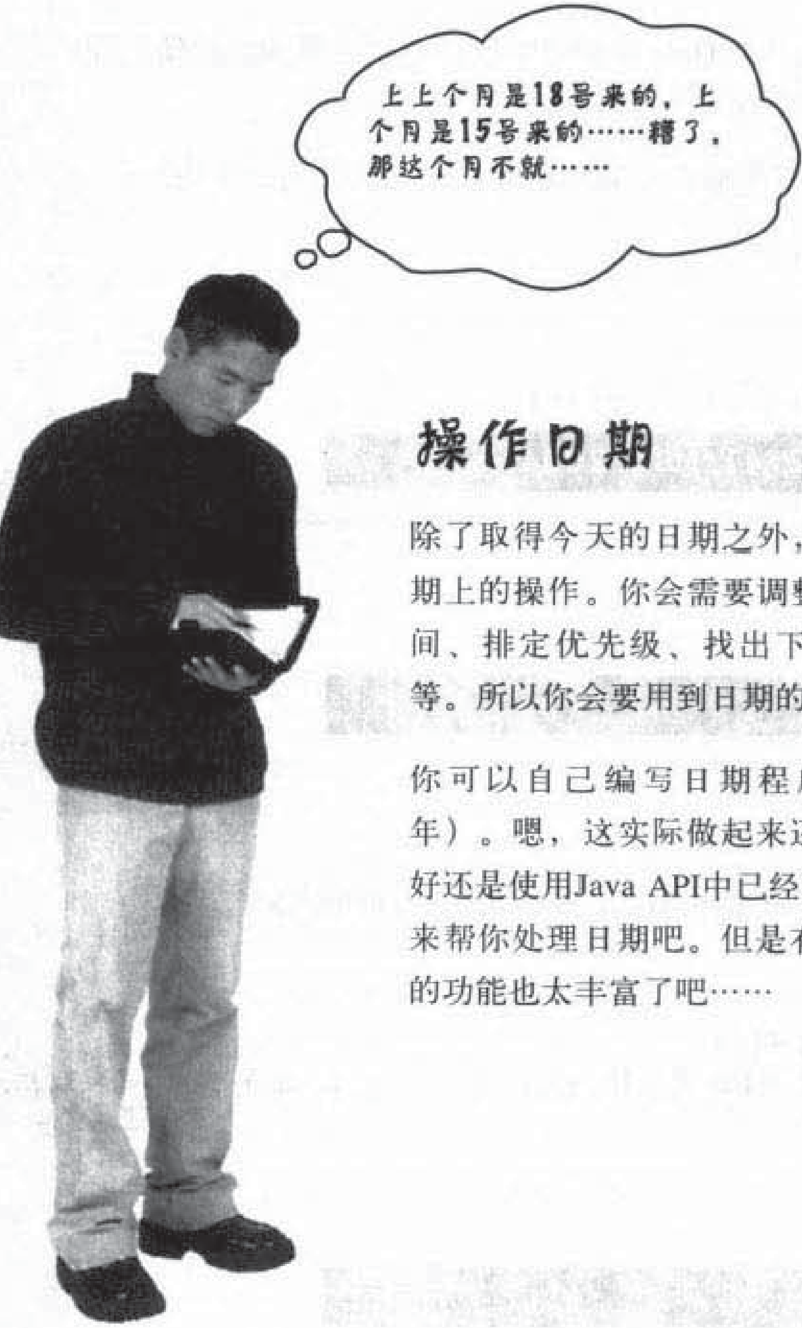
这里的逗号是直接输出的

```
Sunday, November 28
```

同上，但不用重复给参数

```
Date today = new Date();
String.format("%tA, %<tB %<td",today);
```

“<”这个符号是个特殊的指示，用来告诉格式化程序重复利用之前用过的参数



操作日期

除了取得今天的日期之外，你还会遇到很多日期上的操作。你会需要调整日期、计算花费时间、排定优先级、找出下一周期的开始时间等。所以你会要用到日期的高级操作功能。

你可以自己编写日期程序（别忘记处理闰年）。嗯，这实际做起来还蛮复杂的。所以最好还是使用Java API中已经写好、功能丰富的类来帮你处理日期吧。但是有时你会发现这些类的功能也太丰富了吧……

在时光中前后移动

假如说中国厨艺学院的课程表是从周一到周五。你被十八铜人指定要查出今年每个月最后的上课日……

要取得当前的日期时间就用Date，其余功能可以从Calendar上面找。

看来java.util.Date没什么用处

之前我们使用java.util.Date来查询今天的日期，因此从这个类开始寻找适用的功能是很合理的，但当你检查API文件时，你会发现有很多Date的功能被停用了！

但这个类还是很适合用来取得目前的时间。

好消息是API建议到java.util.Calendar上面去寻找其余的功能，因此我们要说：

就用java.util.Calendar来操作日期吧！

Calendar这个API的设计者打算要做全球化的思考。基本的想法是当你要操作日期时，你会要求一个Calendar（通过下一页会讨论的一个静态方法），然后Java虚拟机会赏你一个Calendar的子类实例（实际上Calendar是个抽象的类，所以你能用到的是它的具体子类）。

有意思的是，你所取得的Calendar是符合所在地区（locale）特性的。通常大部分的地区适用公历，但你也有可能处于使用农历或其他特殊格式的情况，此时你可以让Java函数库来处理这一类的日期。

标准的Java API带有java.util.GregorianCalendar，因此我们在书上使用这个历法。通常你也不需要担心你是在使用哪一种Calendar的subclass，只要专注于Calendar的方法就可以了。

上一页的老兄是在看信用卡的账单日期……

取得继承过Calendar的对象

你要如何取得抽象类的“实例”呢？当然不行，下面这个例子根本无法运行。

这个不行：

```
Calendar cal = new Calendar();
```

无法通过编译！

要用这个静态的方法：

```
Calendar cal = Calendar.getInstance();
```

这个语法看起来很熟悉—是个对静态方法的调用





你无法取得Calendar的实例，但是你可以取得它的具体子类的实例

很明显，你不能取得Calendar的实例，因为Calendar是抽象的。但你还是能够不受限制地调用 Calendar 的静态method，这是因为静态的方法是在类上而不是在某个特定的实例上。所以你对Calendar调用getInstance()会返回给你具体子类的实例。那是某种继承过Calendar（也就是Calendar的多态变化版本）并且会依据合约来响应Calendar应有的方法。

大部分的Java版本都会默认返回一个java.util.GregorianCalendar的实例。

运用Calendar对象

要运用Calendar对象你得先了解几个关键的概念：

- 字段会保存状态——Calendar对象使用许多字段来表示某些事物的最终状态，也就是日期和时间。比如说你可以读取和设定它的year或month字段。
- 日期和时间可以运算——Calendar的方法能够让你对不同的字段作加法或减法的运算，比如说对month字段加一个月或对year减去三年。
- 日期与时间可以用millisecond来表示——Calendar可以让你将日期转换成微秒的表示法，或将微秒转换成日期。（更精确的说法是相对于1970年1月1日的微秒数），因此你可以执行精确的相对时间计算。

运用Calendar对象的范例：

```
Calendar c = Calendar.getInstance();
c.set(2004,1,7,15,40);          // 将时间设定为2004年1月7日15:40，注意月份是零基的
long day1 = c.getTimeInMillis(); // 将目前时间转换为以millisecond表示
day1 += 1000 * 60 * 60;
c.setTimeInMillis(day1);        // 将c的时间加上一个小时
System.out.println("new hour " + c.get(c.HOUR_OF_DAY));
c.add(c.DATE, 35);              // 加上35天，所以c已经到了2月
System.out.println("add 35 days " + c.getTime());
c.roll(c.DATE, 35);             // 滚动35天，注意：只有日期字段会动，月份不会动
System.out.println("roll 35 days " + c.getTime());
c.set(c.DATE, 1);               // 直接设定DATE的值
System.out.println("set to 1 " + c.getTime());
```

```
File Edit Window Help Time-Flies
new hour 16
add 35 days Wed Feb 11 16:40:41 MST 2004
roll 35 days Tue Feb 17 16:40:41 MST 2004
set to 1 Sun Feb 01 16:40:41 MST 2004
```

输出结果

Calendar API的精华

这里只有列出Calendar部分的字段和方法。它是相当大的API，因此这里只能列出常用的部分，一旦你熟悉它的操作之后，其余的部分也会很容易使用。

重要的方法

add(int field, int amount)
加减时间值

get(int field)
取出指定字段的值

getInstance()
返回Calendar，可指定地区

getTimeInMillis()
以毫秒返回时间

roll(int field, boolean up)
加减时间值，不进位

set(int field, int value)
设定指定字段的值

set(year, month, day, hour, minute)
设定完整的时间

setTimeInMillis(long millis)
以毫秒指定时间

// 不只这些……

关键字段

DATE / DAY_OF_MONTH
每月的几号

HOUR / HOUR_OF_DAY
小时

MILLISECOND
毫秒

MINUTE
分钟

MONTH
月份

YEAR
年份

ZONE_OFFSET
时区位移

// 还有更多……

PDG

静到最高点！静态的import

这是Java 5.0的新功能：一把双刃剑。有些人很喜欢这个主意，有些人恨死它了。如果你讨厌多打几个字，那你会喜欢这项功能。但它的缺点是会让程序比较难阅读。

基本上，这功能是让你import静态的类、变量或enum（稍后介绍）时能够少打几个字。

警告：这会让你的程序更易混淆

旧式的写法：

```
import java.lang.Math;

class NoStatic {

  public static void main(String [] args) {

    System.out.println("sqrt" + Math.sqrt(2.0));

    System.out.println("tan" + Math.tan(60));

  }

}
```

使用static import的写法：

静态import的语法

```
import static java.lang.System.out;

import static java.lang.Math.*;

class WithStatic {

  public static void main(String [] args) {

    out.println("sqrt" + sqrt(2.0));

    out.println("tan" + tan(60));

  }

}
```

只不过是少打一些字

时机与要领

- 如果只会用到一两次，不如不用静态的import，这样程序会比较好阅读。
- 如果会用到很多次，或许用static的import会让程序看起来比较清爽。
- 在静态import的声明中也可以使用.*这样的通用字符。
- 最重要的问题是很容易产生名称的冲突。例如add()到底是要调用哪个的方法？



今晚的话题：
实例变量对静态变量的卑劣指控

实例变量

静态变量

实例变量： 我不明白为什么要谈这些事情。每个人都知道静态变量只是用来当常数而已。又没有多少常数要用，我猜整个API大约只有四五个吧？真的用过的人也没几个吧。

静态变量： 老兄，没有常识也应该要多看电视。你一定很少看API吧？里面有一堆的静态变量呢。甚至有一整个类是专门用来放常数值的。例如说SwingConstants就全部都是。

实例变量： 对啦，全部都是。不过每个人都知道Swing只是个特殊的例子。

静态变量： 或许它是个特例，但也是个很重要的特例啊！还有Color这个类，如果你记住所有颜色的RGB值那会有多恐怖？所以它已经定义好了红橙黄绿等颜色，用起来很方便的。

实例变量： 是呀，除了少数几个GUI之外，你还能举出任何一个真的大家都在用的静态变量吗？

静态变量： 例如System.out，它是System这个类的一个静态变量。你不必自己创建出System的实例，只要从该class调用out变量就行。

实例变量： 呃……这也算特殊的例子呀。何况大家只是用它来除错而已。

静态变量： 怎样？除错不重要吗？你的豆腐脑一定不知道静态变量比较有效率，共享的类会省下很多内存的！

实例变量 / **静态变量**

实例变量： 嗯……你该不会忘记了吧？

静态变量： 什么？

实例变量： 静态变量不是面向对象的！我们干脆回到古代用算盘来执行计算的工作好了。

静态变量： 别乱扣帽子，不是面向对象怎么了？那是什么？

实例变量： 就像全局变量一样，许多潇洒帅气的程序员都知道那通常是很负面的事情。

静态变量： 对不起，我并不是全局变量。根本没有全局变量，我是在面向对象化的类中的！我是对象的自然状态，唯一的差别是我被很有效率地共享。

实例变量： 这样就不叫面向对象程序设计了，这叫笨蛋。你真的是老古董啊。

静态变量： 够了！闭嘴！这不是真的。某些静态变量对系统是非常关键的，不然至少也能够提供便利性。

实例变量： 对啦，偶尔用一下静态变量还算合理，但是我要告诉你，滥用静态变量和方法就是不成熟面向对象程序员的招牌。程序员应该想的是对象的状态而不是类的状态。

静态变量： 你说什么？这又跟静态的方法扯上什么关系？

实例变量： 这代表程序员还在用程序化的思维想事情，而不是依据对象的状态来进行处理。

静态变量： 当然，我同意面向对象设计应该要专注于对象，有些问题只是因为菜鸟的因素……当你真的有需要的时候，没有东西能够代替静态。

实例变量： 是是是，就你自己需要嘛！



我是编译器

这一页的Java程序代码代表一份完整的程序。你的任务是要扮演编译器角色并判断程序输出会是哪一个？



```
class StaticSuper{

  static {
    System.out.println(“super static block”);
  }

  StaticSuper{
    System.out.println(
      “super constructor”);
  }
}

public class StaticTests extends StaticSuper {
  static int rand;

  static {
    rand = (int) (Math.random() * 6);
    System.out.println(“static block ” + rand);
   }

   StaticTests() {
     System.out.println(“constructor”);
   }

  public static void main(String [] args) {
    System.out.println(“in main”);
    StaticTests st = new StaticTests();
  }
}
```

哪个才是它的输出？

可能输出：

```
File Edit Window Help Cling
%java StaticTests
static block 4
in main
super static block
super constructor
constructor
```

可能输出：

```
File Edit Window Help Electricity
%java StaticTests
super static block
static block 3
in main
super constructor
constructor
```



这一章讨论 Java 的静态世界。你的任务是辨别下面的陈述哪些是对的、哪些是错的？

是非题

（1）使用Math类的第一个步骤是创建出它的实例。

（2）构造函数可以标记为静态的。

（3）静态的方法不能存取“this”所引用的对象。

（4）最好通过引用变量来调用静态的方法。

（5）静态变量可以用来计算类的实例数量。

（6）构造函数是在静态变量的初始化之前执行的。

（7）MAX_SIZE是个合法的静态final变量名称。

（8）静态初始化程序会在构造函数之前执行。

（9）如果类被标记为final，则它的方法也必须标记为final。

（10）final的方法只能在它的类被继承时覆盖。

（11）boolean类型没有包装用的类。

（12）wrapper是用来把primitive主数据类型包装成对象。

（13）parseXxx方法都会返回String。

（14）格式化的类都在java.format中包。



排排看

下面是被打散的Java程序片段，程序的目的是要以29.52的周期计算满月；上一次的满月是在2004年1月7日。你是否能够将它们重新排列以成为可以编译并执行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
long day1 = c.getTimeInMillis();
```

```
c.set(2004,1,7,15,40);
```

```
import static java.lang.System.out;
```

```
static int DAY_IM = 60 * 60 * 24;
```

```
("full moon on %tc", c));
```

```
(c.format
```

```
Calendar c = new Calendar();
```

```
class FullMoons {
```

```
public static void main(String [] args) {
```

```
day1 += (DAY_IM * 29.52);
```

```
for (int x = 0; x < 60; x++) {
```

```
static int DAY_IM = 1000 * 60 * 60 * 24;
```

```
println
```

```
import java.io.*;
```

```
("full moon on %t", c));
```

```
import java.util.*;
```

```
static import java.lang.System.out;
```

```
c.set(2004,0,7,15,40);
```

```
out.println
```

```
c.setTimeInMillis(day1);
```

```
(String.format
```

```
Calendar c = Calendar.getInstance();
```

File Edit Window Help Howl

```
% java FullMoons
full moon on Fri Feb 06 04:09:35 MST 2004
full moon on Sat Mar 06 16:38:23 MST 2004
full moon on Mon Apr 05 06:07:11 MDT 2004
```

练习解答

我是编译器

```
StaticSuper( ) {
  System.out.println(
    "super constructor");
}
```

构造函数必须要有括号。

可能输出

```
File Edit Window Help Cling
%java StaticTests
super static block
static block 3
in main
super constructor
constructor
```

是非题

(1) 错

(2) 错

(3) 对

(4) 错

(5) 对

(6) 错

(7) 对

(8) 对

(9) 错

(10) 错

(11) 错

(12) 对

(13) 错

(14) 错



注意：这个解决方案有些取巧的做法，如果要做到完美就又太过复杂了。

```java
import java.util.*;
import static java.lang.System.out;
class FullMoons {
  static int DAY_IM = 1000 * 60 * 60 * 24;
  public static void main(String [] args) {
    Calendar c = Calendar.getInstance();
    c.set(2004,0,7,15,40);
    long day1 = c.getTimeInMillis();
    for (int x = 0; x < 60; x++) {
       day1 += (DAY_IM * 29.52)
       c.setTimeInMillis(day1);
       out.println(String.format("full moon on %tc", c));
    }
  }
}
```

```
File Edit Window Help Howl
% java FullMoons
full moon on Fri Feb 06 04:09:35 MST 2004
full moon on Sat Mar 06 16:38:23 MST 2004
full moon on Mon Apr 05 06:07:11 MDT 2004
```

11 异常处理

有风险的行为



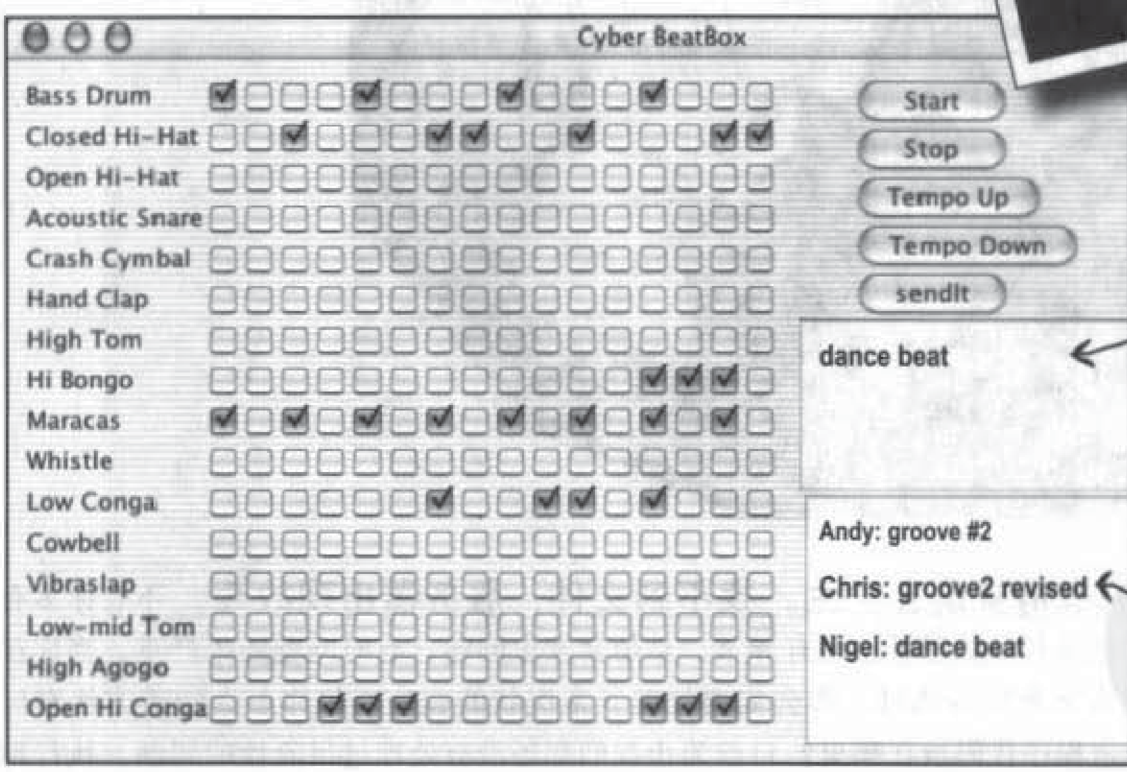
倒霉的事情就是会发生。找不到文件、服务器出现故障。不管你多有天份，你也没有办法保证不会有异常。总是会发生问题，有时问题还很严重。当你在编写可能有异常的方法时，你会需要处理异常况状的程序。但你怎么知道方法有风险呢，异常程序代码放在哪里？目前为止我们都还没有处理过风险性的问题。执行期肯定会有问题，但大多数都来自于程序的错误，而这些错误应该在开发阶段解决掉。不过我们所说的问题是你无法保证在执行期不会出现的。例如预期某些文件会正确地待在某个特定的目录中，但实际执行时文件却又失踪了。这一章我们会使用具有风险的JavaSound API来创建一个MIDI音乐播放程序。

创建MIDI音乐播放器



我们会在接下来的三章创建一个不太一样的声效应用程序，其中包括了BeatBox Drum播放机。在本书结束之前，我们会有个可以把鼓声送给另外一个播放器的版本，这有点类似讨论版。你可以全部自己动手写，也可以下载已经写好的GUI部分。虽然这个程序没什么商业价值，但是可以通过这个过程学习Java。这样的过程只是学习Java的一种比较有趣的方式。

对checkbox打勾来设定是否要发出声音



按下sendit就会将此信息连同目前的节拍组合一起发送

来自其他人的信息，点选就会加载它的节拍组合

你可以在上面的16个拍子上打勾。例如在第一拍Bass Drum与Maracas会发出声音，而第二拍不会有声音，Maracas与Closed Hi-Hat会在第三拍发声……这样懂了吧？按下Start会持续循环地演奏，按Stop就会停下来。任何时候你都可以送出目前的组合到BeatBox服务器上，让其他人也能够聆听你的经典大作。你也可以点选别人送来的信息以加载节拍组合。

从基本谈起
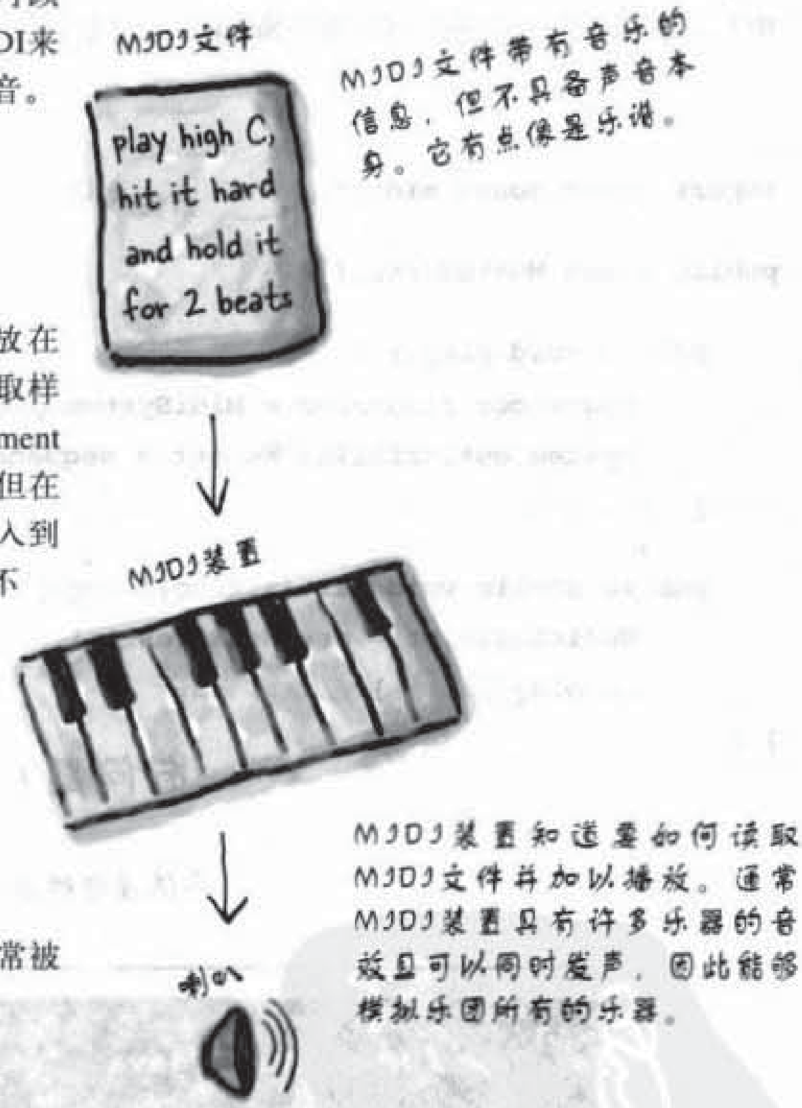
很明显的，我们得要先学习某些事情才能完成这个程序。这包括了如何创建Swing GUI、如何通过网络连接到其他计算机，以及让我们可以把数据送出的输入/输出设备。

当然还有JavaSound这个API。这一章会先谈这个部分，你可以先暂时忘掉GUI与网络联机等部分，现在只要专注于由MIDI来发出声音就好。这一章会说明MIDI以及如何读取与播放声音。你是否已经开始想象上台领金曲奖的模样？

JavaSound API

这是在Java 1.3之后所加入的一组类和接口。它们是放在J2SE 的类函数库中。JavaSound被分为两个部分：MIDI和取样（sampled）。这本书只会讨论MIDI，它代表Musical Instrument Digital Interface，也是不同电子发声装置沟通的标准协议。但在我们的程序中，你可以把MIDI想象成某种乐谱，它可以输入到某种“高级多功能电子魔音琴”中。换句话说，MIDI本身不带有声音，它带的是有MIDI播放功能装置的指令。

MIDI数据表示执行的动作（播放中央C，以及音量大小和长度等），但没有实际的声音。它并不知道怎样产生钢琴、吉它、鼓的音效，实际的声音是靠装置发出的。这样的装置很像是个编制完整的交响乐团，它可能是个高级键盘乐器，也有可能是计算机中的纯软件音源。



对BeatBox来说，只会使用内建、纯软件的乐器音效。这通常被称为synthesizer（有些人把它叫做software synth）。

首先我们需要一个Sequencer

在我们要能够发出任何声音之前，必须先要取得Sequencer对象。此对象会将所有的MIDI数据送到正确的装置上，由装置来产生音乐。sequencer实际上可以做很多事情，但在我们的章节中所指的是播放的装置，就好像是你家中音响的CD唱盘Sequencer这个类位于javax.sound.midi这个包中（Java 1.3版之后的标准Java函数库中），所以我们先从确认可以取得Sequencer对象开始。

```
import javax.sound.midi.*;

public class MusicTest1 {

    public void play() {
        Sequencer sequencer = MidiSystem.getSequencer();
        System.out.println("We got a sequencer");
    } // 关闭播放

    public static void main(String[] args) {
        MusicTest1 mt = new MusicTest1();
        mt.play();    } // 关闭 main
} // 关闭类
```

引用该包

这个对象的作用是将MIDI信息组合成"乐曲"。

有问题！

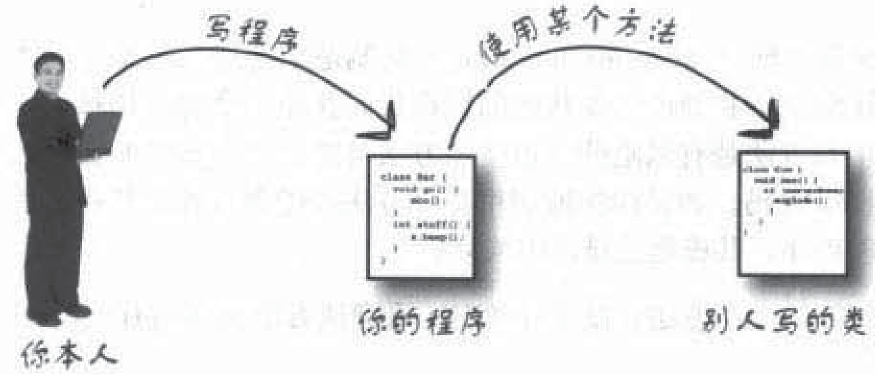
无法通过编译！编译器表示有异常状况必须要处理

```
File Edit Window Help SayWhat?

% javac MusicTest1.java
MusicTest1.java:13: unreported exception javax.sound.midi.
MidiUnavailableException; must be caught or declared to be
thrown
    Sequencer sequencer = MidiSystem.getSequencer();
                                                 ^
1 errors
```

调用有风险的方法（或许不是你自己写的）时会发生什么事？

① 假设你调用了一个不是自己写的方法。

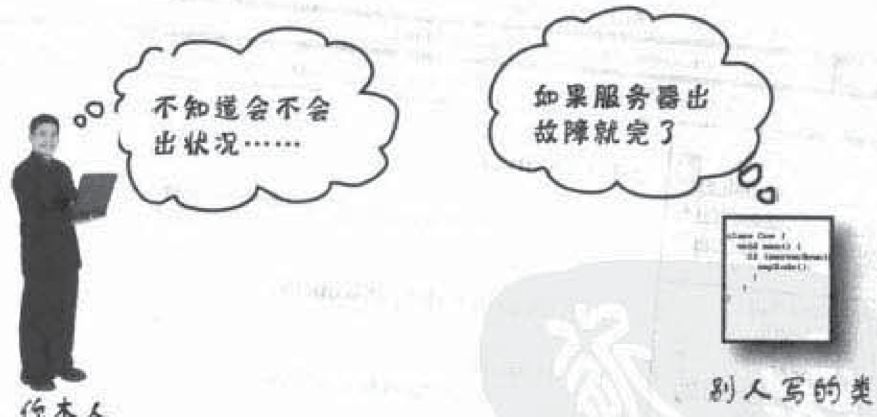


② 该方法执行某些有风险的任务，可能会在运行期间出状况。




```
void moo() {
   if (serverDown) {
    explode();
   }
}
```

③ 你必须认识到该方法是有风险的。



④ 你得写出可以在发生状况时加以处理的程序代码，未雨绸缪！

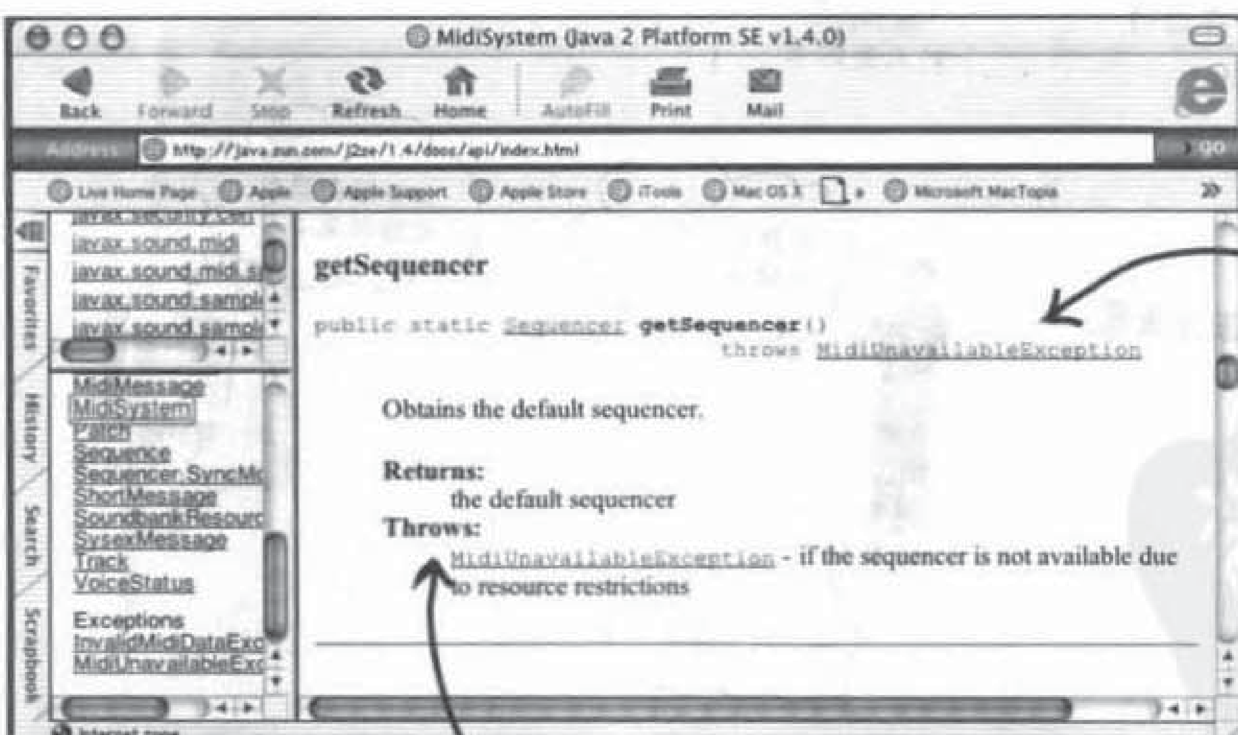


Java使用异常来告诉调用方法的程序代码："有问题！我不行了！"

Java的异常处理（exception-handling）机制是个简捷、轻量化的执行期间例外状况处理方式，它让你能够将处理错误状况的程序代码摆在一个容易阅读的位置。这要依赖你已经知道所调用的方法是有风险的（也就是方法可能会产生异常），因此你可以编写出处理此可能性的程序代码。如果你知道调用某个方法可能会有异常状况，你就可以预先准备好对问题的处理程序，甚或是从错误中恢复。

你要如何得知某个方法会抛出异常呢？看到该方法的声明有throws语句就知道了。

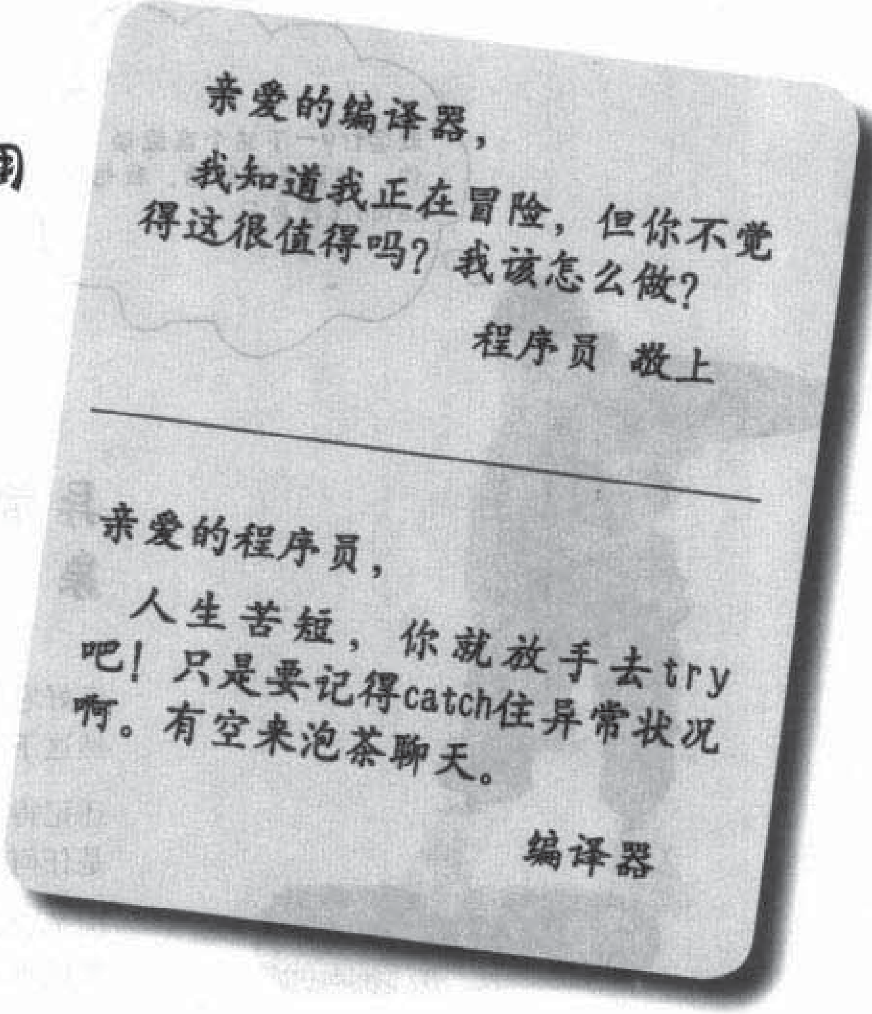
getSequencer()这个方法可能会在执行期间出问题，所以必须声明出调用它可能会有风险。



API文件说明getSequencer()可能会抛出MidiUnavailableException异常

此处解释何时会遇到异常，在此情况下是因为资源受限，比如sequencer已经被占用了

编译器要确定你了解所调用的方法是有风险的



如果你把有风险的程序代码包含在try/catch块中，那么编译器会放心许多。

try/catch块会告诉编译器你确实知道所调用的方法会有风险，并且也已经准备好要处理它，它只会注意你有没有表示你会注意到异常。

```
import javax.sound.midi.*;

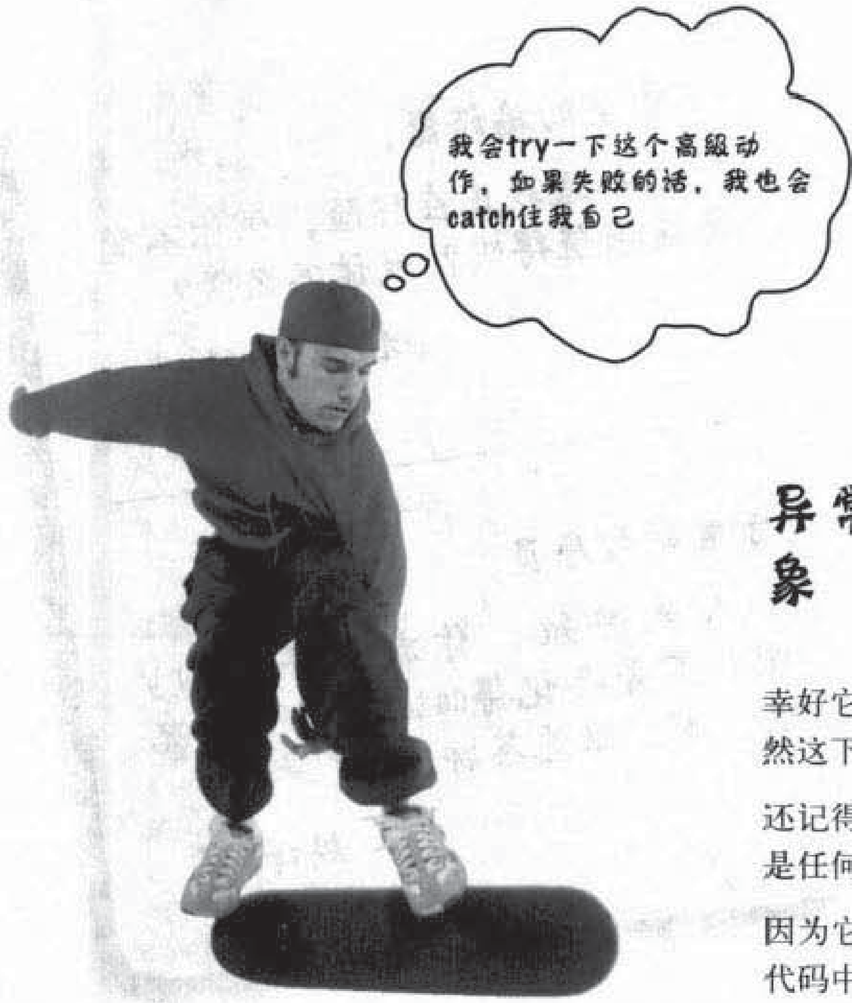
public class MusicTest1 {
    public void play() {

        try {
            Sequencer sequencer = MidiSystem.getSequencer();
            System.out.println("Successfully got a sequencer");
        } catch(MidiUnavailableException ex) {
            System.out.println("Bummer");
        }
    } // 关闭播放

    public static void main(String[] args) {
        MusicTest1 mt = new MusicTest1();
        mt.play();
    } // close main
} // 关闭类
```

把有风险的程序放在try块

用catch块摆放异常状况的处理程序

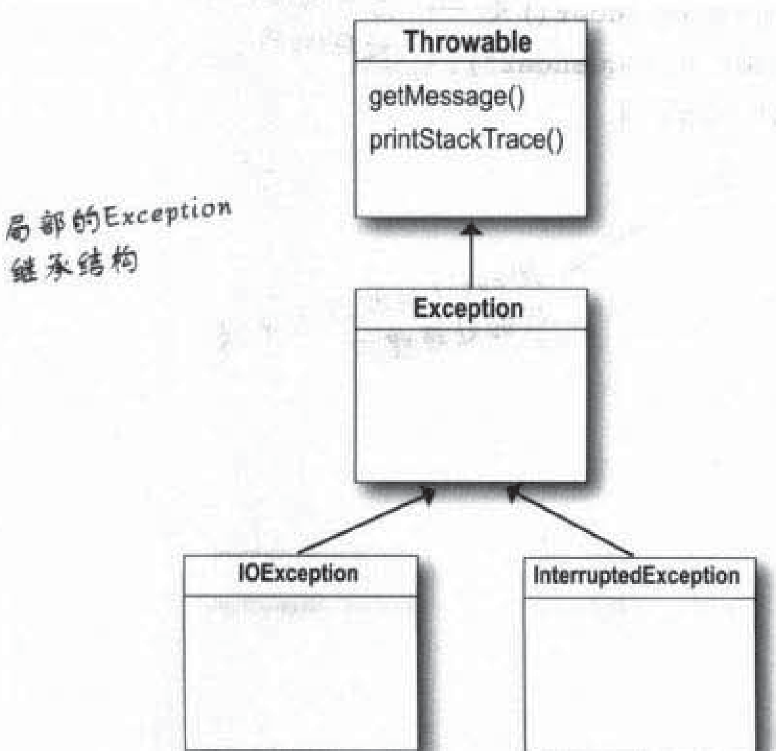


异常是一种Exception类型的对象

幸好它的类型名称不是Asabulu543AlubaE04Aligado，不然这下可难记了。

还记得关于多态的那一章提到Exception类型的对象可以是任何它的子类的实例吗？

因为它是对象，所以你catch住的也是对象。下面的程序代码中catch的参数是Exception类型的ex引用变量：




```
try {
    // 危险动作

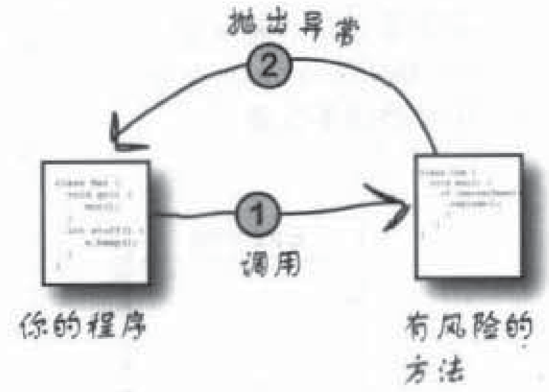
} catch(Exception ex) {
    // 尝试恢复
}
```

就跟方法的参数声明一样

这一段只会在有抛出异常时执行

你写在catch块中的程序必定与所抛出的异常有关。例如，如果遇到服务器出故障的异常，你可能会在这里写寻找替代服务器的方法。

如果你的程序代码会抓住异常，那是谁把它抛出来的？



你花在处理异常的程序设计时间会比花在自己创建与抛出异常的时间还多很多。现在只要知道当你的程序代码调用有风险的方法时，也就是声明有异常的方法，就是该方法把异常丢给你的。

实际上，两者可能都是你自己写的。由谁写的程序其实并不重要，重点在于哪个方法抛出异常与哪个方法抓到它。

在编写可能会抛出异常的方法时，它们都必须声明有异常。

① 有风险、会抛出异常的程序代码：

```
public void takeRisk() throws BadException {
  if (abandonAllHope) {
    throw new BadException();
  }
}
```

必须要声明它会抛出BadExcecption

创建Exception对象并抛出

方法可以抓住其他方法所抛出的异常。异常总是会丢回给调用方。

会抛出异常的方法必须要声明它有可能会这么做。

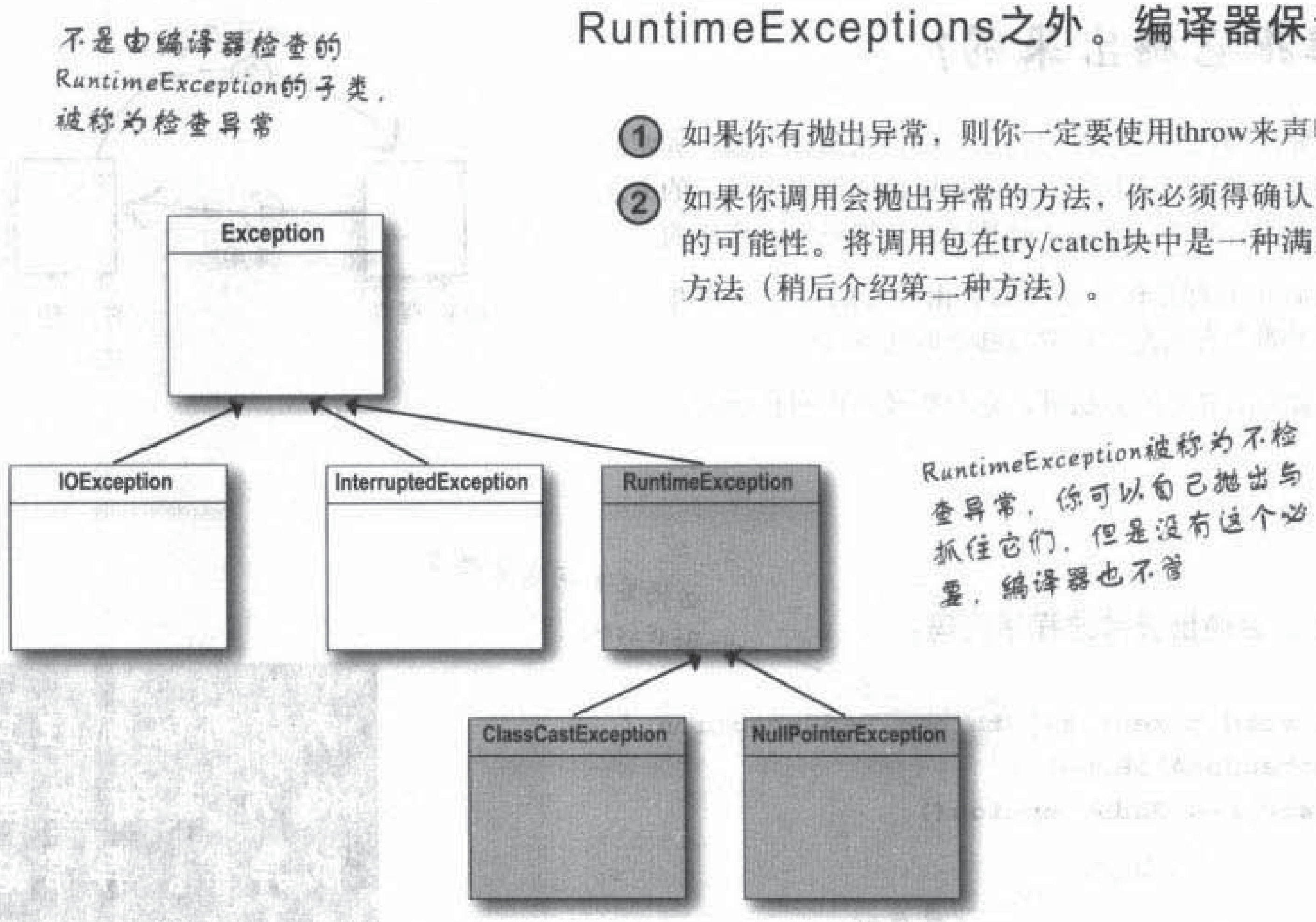
② 调用该方法的程序代码：

```
public void crossFingers() {
  try {
    anObject.takeRisk();
  } catch (BadException ex) {
    System.out.println("Aaargh!");
    ex.printStackTrace();
      }
}
```

如果无法从异常中恢复，至少也要使用printStackTrace()来列出有用的信息

编译器会核对每件事，除了RuntimeExceptions之外。编译器保证：

1. 如果你有抛出异常，则你一定要使用throw来声明这件事。
2. 如果你调用会抛出异常的方法，你必须得确认你知道异常的可能性。将调用包在try/catch块中是一种满足编译器的方法（稍后介绍第二种方法）。



there are no Dumb Questions

问： 等一下！这怎么会是我们第一次遇到必须处理的异常？我已经遇过NullPointerException和DivideByZeror，甚至Integer.parseInt()也关照我NumberFormatException，为什么这些都不用处理？

答： 除了RuntimeException这种特例之外，编译器会关照Exception所有的子类。任何继承过RuntimeException的类都不会受编译器关于是否声明它会抛出RuntimeException的检查，同样的，也不会管调用方是否认识到可能会在运行期间遇到异常。

问： 为什么编译器不管那些运行期间的异常？它们不也是会让整个程序跟着死掉吗？

答： 大部分的RuntimeException都是因为程序逻辑的问题，而不是以你所无法预测或防止的方法出现的执行期失败状况，你无法保证文件一直都在。你无法保证服务器不会死机。但是你可以确保程序不会运行不合理的逻辑，例如对只有5项元素的数组取第八个元素的值。

你会需要在开发与测试期间发生RuntimeException，以便能够在不把程序代码放进try/catch块的情况下来抓一开始就不应该出现的问题。

try/catch是用来处理真正的异常，而不是你程序的逻辑错误，该块要做的是恢复的尝试，或者至少会优雅的列出错误信息。

要点

- 方法可在运行期间遇到问题时抛出异常。
- 异常是Exception类型的对象。
- 编译器不会注意RuntimeException类型的异常。RuntimeException不需要声明或被包在try/catch的块中（然而你还是可以这么做）。
- 编译器所关心的是称为检查异常的异常。程序必须要认识有异常可能的存在。
- 方法可以用throw关键词抛出异常对象：

```
throw new FileIsTooSmallException( );
```

- 可能会抛出异常的方法必须声明成throws Exception。
- 如果程序调用了有声明会抛出异常的方法，就得要告诉编译器已经注意到这件事。
- 如果要处理异常状况，就把调用包在try/catch块中，并将异常处理/恢复程序放在catch块中。
- 如果不打算处理异常，还是可以正式地将异常给ducking来通过编译，稍后会解释ducking。

metacognitive tip

如果想要学习某些事物，就把它当作是睡前最后学习的东西。当你放下书本时，就不要再做其他需要动脑的事情。大脑需要时间处理学习内容。这可能要几个钟头，如果此时有还有别的事情要想，之前所看过的东西可能会被遗忘。

当然啦，这跟技巧性的活动无关，或许平日的自由搏击或摔跤练习并不会影响到你对Java的学习。

最好的方法是在就寝前看这本书（就算只是看看图片也无所谓）。



Sharpen your pencil

右列有哪些事情是你认为编译器会在乎的异常？我们只要找出无法在程序中控制的事情。第一项已经帮你写好了（因为它是最简单的）。

要执行的工作	会有什么问题？
✔ 连接远程服务器	服务器死掉
__ 存取数组	
__ 显示Window	
__ 从数据库取出数据	
__ 判断文件是否存在	
__ 打开文件	
__ 从命令列读取字符	

try/catch块的流程控制

当你要调用有风险的方法时，有一两件事情可能会发生。该方法若不是成功地把try块完成的话，不然就是会把异常丢回调用方的方法。

如果成功的话：

```
try {
  ① Foo f = x.doRiskyThing();
    int b = f.getNum();

} catch (Exception ex) {
    System.out.println("failed");
}
② System.out.println("We made it!");
```

try块会先运行，然后会跳过catch块运行下面的部分

这段不会运行

```
File Edit Window Help RiskAll
%java Tester
We made it!
```

如果失败：

```
try {
  ① Foo f = x.doRiskyThing();
    int b = f.getNum();

} catch (Exception ex) {
  ② System.out.println("failed");
}
③ System.out.println("We made it!");
```

调用doRiskyThing()抛出异常，所以会跳过其余的程序运行到catch块执行，然后继续下去

其余的部分不会运行，这样很好，因为那些都要依赖之前的成功才能执行下去

```
File Edit Window Help RiskAll
%java Tester
failed
We made it!
```

finally：无论如何都要执行的部分

如果你要煮东西吃，得先把炉子打开。

如果你的烹饪过程失败了，必须把炉子关掉。

如果你成功了，必须把炉子关掉。

不管怎样，你终究得关掉炉子。

finally块是用来存放不管有没有异常都得执行的程序。

```
try {
  turnOvenOn();
  x.bake();
} catch (BakingException ex) {
    ex.printStackTrace();
} finally {
    turnOvenOff();
}
```

如果没有finally，你得同时把turnOvenOff()摆在try与catch两处。finally块可以让你把所有重要的清理程序代码集中在一处，而不需要复制两份成下面这样：

```
try {
  turnOvenOn();
  x.bake();
  turnOvenOff();
} catch (BakingException ex) {
    ex.printStackTrace();
    turnOvenOff();
}
```



如果try块失败了，抛出异常，流程会马上转移到catch块。当catch块完成时，会执行finally块。当finally完成时，就会继续执行其余的部分。

如果try块成功，流程会跳过catch块并移动到finally块，当finally完成时，就会继续执行其余的部分。

如果try或catch块有return指令，finally还是会执行！流程会跳到finally然后再回到return指令。

Sharpen your pencil

流程控制

假设ScaryException继承Exception，观察左方的程序代码。你认为这个程序会有怎样的输出？如果程序的第三行改成：String test = “yes”，会怎样？

```
public class TestExceptions {

  public static void main(String [] args) {

    String test = "no";
    try {
       System.out.println("start try");
       doRisky(test);
       System.out.println("end try");
    } catch ( ScaryException se) {
        System.out.println("scary exception");
    } finally {
        System.out.println("finally");
    }
    System.out.println("end of main");
  }

  static void doRisky(String test) throws ScaryException {
    System.out.println("start risky");
    if ("yes".equals(test)) {

      throw new ScaryException();
    }
    System.out.println("end risky");
    return;
  }
}
```

输出当test = "no"

输出当test = "yes"

When test = "no": start try - start risky - end risky - end try - finally - end of main

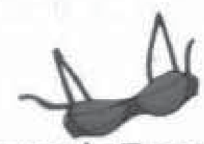
When test = "yes": start try - start risky - scary exception - finally - end of main

我们讨论过方法可以抛出一个以上的异常吗？

如果有必要的话，方法可以抛出多个异常。但该方法的声明必须要有含有全部可能的检查异常（若两个或两个以上的异常有共同的父类时，可以只声明该父类就行）。

处理多重异常

编译器会检查你是否处理所有可能的异常。将个别的catch块逐个放在try块下。某些情况下catch出现的先后顺序会有影响，但这部分我们稍后再加以说明。

```
public class Laundry {
    public void doLaundry() throws PantsException, LingerieException {
        // 有可能抛出两个异常的程序代码
    }
}
```

声明两个可能的异常类型



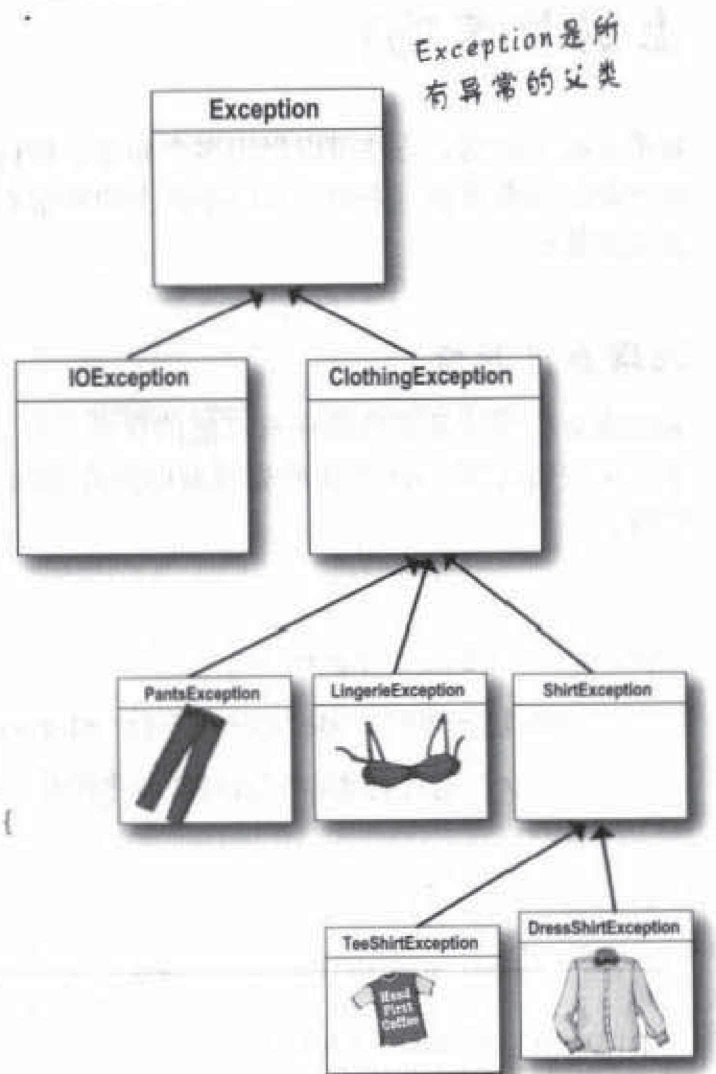

```
public class Foo {
    public void go() {
        Laundry laundry = new Laundry();
        try {
            laundry.doLaundry();
        } catch(PantsException pex) {
            // 恢复程序代码
        } catch(LingerieException lex) {
            // 恢复程序代码
        }
    }
}
```

如果抛出的是PantsException，它就会运行到这个块

如果抛出的是LingerieException，则跳到这个段

异常也是多态的

别忘记异常是对象。除了可以被抛出之外，并没有什么特别的。因此如同所有的对象，异常也能够以多态的方式来引用。举例来说，LingerieException对象能被赋值给ClothingException的引用。PantsException也能够被赋值给Exception的引用。这样的好处是方法可以不必明确地声明每个可能抛出的异常，可以只声明父类就行。对于catch块来说，也可以不用对每个可能的异常作处理，只要有一个或少数几个catch 可以处理所有的异常就够了。



① 以异常的父型来声明会抛出的异常



```
public void doLaundry() throws ClothingException {
```

声明成ClothingException可让你抛出任何ClothingException的子类。这代表此方法可以抛出PantsException、LingerieException等异常而不用个别的声明

② 以所抛出的异常父型来catch异常

```
try {
    laundry.doLaundry();
```



```
} catch(ClothingException cex) {
    // 解决方案
}
```

可以catch任何ClothingException的子类

```
try {
    laundry.doLaundry();
```



```
} catch(ShirtException sex) {
    // 解决方案
}
```

只能catch其两种子类

可以用super来处理所有异常并不代表就应该这么做

你可以把异常处理程序代码写成只有一个catch块以父型的Exception来捕获，因此就可以抓到任何可能被抛出的异常：

```
try {
    laundry.doLaundry();
} catch(Exception ex) {
    // 解决方案……
}
```

恢复什么？这会捕获所有的异常，因此你会搞不清楚哪里出错

为每个需要单独处理的异常编写不同的catch块

举例来说，如果你的程序代码处理TeeShirException的方法与LigngerieException的方法不同，则要个别地写出catch块。但如果ClothingException都是以同样的方式处理，则可以使用ClothingException的catch来处理。

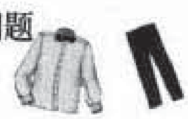
```
try {
    laundry.doLaundry();
```



```
} catch(TeeShirtException tex) {
    // 恢复此问题
```



```
} catch(LingerieException lex) {
    // 恢复此问题
```



```
} catch(ClothingException cex) {
    // 恢复其他问题
}
```

两者的处理不同，所以使用不同的块

同样处理方法的都会在这边

有多个catch块时要从小排到大



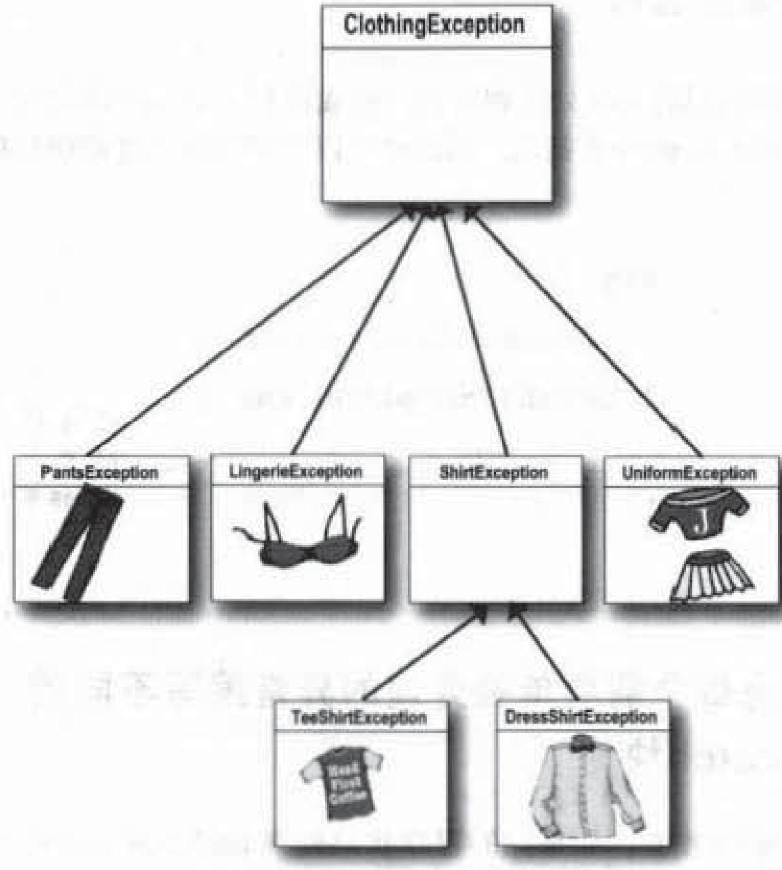
`catch(TeeShirtException tex)`



`catch(ShirtException sex)`



`catch(ClothingException cex)`



在继承树上层次越高，则“篮子”就越大。若你从上往下沿着继承层次走，异常类就会越来越有特定的取向，且catch的篮子也会越来越小。这是多态的常态现象。

ShirtException足以容下TeeShirtException或DressShirtException。而ClothingException甚至更大（能够引用的范围更多），但真的要说到大，Exception类型无疑是头号的霸王，它可以catch所有的异常，还包括了运行期间（unchecked）的异常，因此你或许不会把它用在测试以外的环境中。

不能把大篮子放在小篮子上面

嗯，你硬要这么做也可以，但是会无法通过编译。catch块不像重载的方法会被挑出最符合的项目。使用catch块时，Java虚拟机只会从头开始往下找到第一个符合范围的异常处理块。如果第一个catch就是catch(Exception ex)，则编译器会知道其余的都没有用处——绝对不会被用到。

亲爱的，大小很重要噢！小的要先上，不然就根本不会有机会使用

别这样！




```
try {
  laundry.doLaundry();
} catch(ClothingException cex) {
  // 恢复此问题
```



```
} catch(LingerieException lex) {
  // 恢复此问题
```



```
} catch(ShirtException sex) {
  // 恢复此问题
}
```



姐妹可以一起上……次序不重要，因为两者不会吞下对方的异常

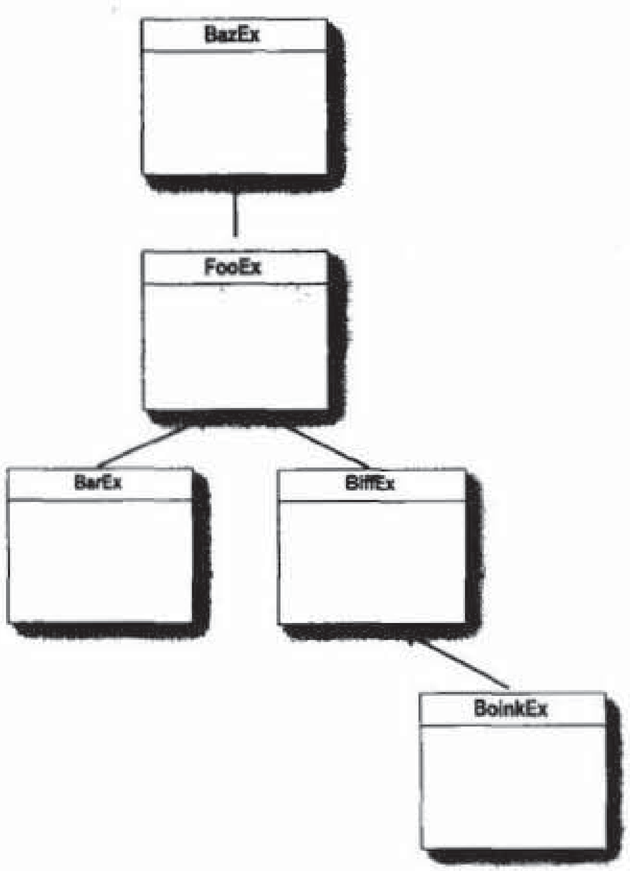
把ShirtException放在LingerieException上面不会有问题，因为ShirtException不会catch到LingerieException这个异常。

Sharpen your pencil

假设左方的块是合法的程序。你的任务是画两个不同的类图来精确地反映出Exception。换句话说，哪种类继承结构能够使这段程序代码为合法的？

```
try {
  x.doRisky();
}  catch(AlphaEx a) {
     // 恢复此问题
}  catch(BetaEx b) {
   // 恢复此问题
}  catch(GammaEx c) {
     // 恢复此问题
}  catch(DeltaEx d) {
     // 恢复此问题
}
```

你的工作是创建出两个不同的try/catch块结构（类似左上），以精确地展现左边的类图。假设所有的异常都会被方法中的try块所抛出。



不想处理异常时……

just duck it



如果不想处理异常，你可以把它duck掉来避开。

当你调用有危险的方法时，编译器需要你对这件事情有所表示。大部分情况下这代表说得把此调用包在try/catch块中。但也可以实行不同的方案：把它duck掉以让调用你的方法的程序来catch该异常。

这很容易，你只要表示出你会再throw此异常就好。技术上，其实它也不是你抛出的，不过这不重要。你只是让异常有出路而已，所以发生异常状况时会怎样呢？

方法抛出异常时，方法会从栈上立即被取出，而异常会再度丢给栈上的方法，也就是调用方，如果调用方是个ducker，则此ducker也会从栈被取出，异常再度抛给此时栈上方的方法，如此一路下去。何时会终止？稍后分晓。

```
public void foo() throws ReallyBadException {
    // 调用有风险的方法
    laundry.doLaundry();
}
```

并没有try/catch块来处理有风险的方法，因此这个方法本身就是有风险的

PDG

ducking只是在踢皮球

早晚还是得有人来处理这件事。但若连main()也duck掉异常呢？

```
public class Washer {
    Laundry laundry = new Laundry();

    public void foo() throws ClothingException {
       laundry.doLaundry();
    }

   public static void main (String[] args) throws ClothingException {
       Washer a = new Washer();
       a.foo();
    }
}
```

两者都躲避异常，因此没人来处理，但有duck掉，所以可以通过编译

❶ 抛出ClothingException。



main()调用foo()

foo()调用doLaundry()
doLaundry()抛出ClothingException。

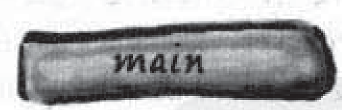
❷ foo()已经duck掉异常。



doLaundry()从stack上被取走。

异常抛给 foo()。

❸ 连main()也duck掉异常。



foo()也被取走……最后只剩下Java虚拟机，你知道这家伙对异常是没有什么责任感的。

❹ Java虚拟机只好死给你看。

我们使用T-Shirt来展示衣服的异常。其实我们知道你最喜欢的是比基尼。

处理或声明，做个堂堂正正的程序员

我们已经看过两种满足编译器的有风险方法调用方式。

● 处理。

把有风险的调用包在try/catch块中

```
try {
    laundry.doLaundry();
} catch(ClothingException cex) {
        // 恢复程序代码
}
```

这最好能处理掉所有doLaundry()可能抛出的异常，不然编译器还是会跟你句句念

● 声明（duck 掉）。

把method声明成跟有风险的调用一样会抛出相同的异常

```
void foo() throws ClothingException {
    laundry.doLaundry();
}
```

有声明会抛出异常，但没有try/catch块，所以就会duck掉异常留给调用方

这代表调用foo()的程序必须要处理或也跟着声明异常

```
public class Washer {
    Laundry laundry = new Laundry();

    public void foo() throws ClothingException {
        laundry.doLaundry();
    }

    public static void main (String[] args) {
        Washer a = new Washer();
        a.foo();
    }
}
```

有问题！无法通过编译，且会有"unreported exception"错误信息

要不就用try/catch块包起来，不然就duck掉

回到音乐播放程序……

你应该已经快要忘记这一章一开始要讲的是JavaSound程序代码。我们把sequencer对象创建出来，但因为Midi.getSequencer()声明了检查异常（MidiUnavailableException）使得程序无法通过编译。现在我们就可以通过包装在try/catch块的调用来解决这个问题。

```
public void play() {
    try {

        Sequencer sequencer = MidiSystem.getSequencer();
        System.out.println("Successfully got a sequencer");

    } catch(MidiUnavailableException ex) {
        System.out.println("Bummer");
    }
} // 关闭播放
```

包在try/catch块中就可以通过编译

catch的参数必须要是正确的异常，你一定要捕获有可能被抛出的异常

异常处理规则

- **catch与finally不能没有try。**

```
void go() {
   Foo f = new Foo();
   f.foof();
   catch(FooException ex) { }
}
```

缺了try!

- **try与catch之间不能有程序。**

```
try {
   x.doStuff();
}
int y = 43;
} catch(Exception ex) { }
```

不能在这里放程序

- **try一定要有catch或finally。**

```
try {
   x.doStuff();
} finally {
   // 清理
}
```

这是合法的，但还是注意第四项

- **只带有finally的try必须要声明异常。**

```
void go() throws FooException {
   try {
      x.doStuff();
   } finally { }
}
```

程序料理



你不一定要亲自下厨，但自己动手的乐趣还是比较多的。

本章接下来的部分是选择性的材料，你也可以下载已经写好的程序代码。

如果想知道更多的细节，就继续读下去吧！

实际发出声音

记得在本章开始时我们已经看过MIDI数据是如何保存应该演奏哪些音乐的指令，并且也了解到MIDI数据其实并没有夹带实际发出的声音。对于真正要给音箱发出的声音而言，MIDI的数据还需送到某种MIDI装置上，并将数据转换成声音。本书只使用软件装置来发声，以下是JavaSound的工作原理：

4项必备的条件：

① 发声的装置

② 要演奏的乐曲

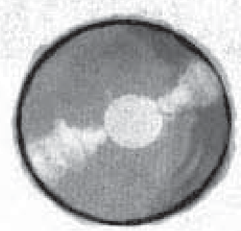
③ 带有乐曲的信息记录

④ 乐曲的音符等信息

Sequencer —plays→ **Sequence** —has a→ **Track** —holds→

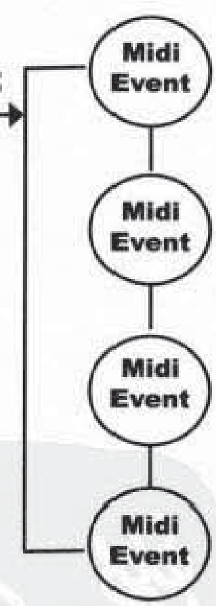


把sequencer想做成CD播放机



sequence就好像是单曲CD

track可比喻是单曲CD上唯一歌曲的信息



可被唱盘理解的信息数据，MIDI event就好像乐谱上的某一个音符记号，也可以用来表示更换乐器的指令等。

音符等指令组成一条歌，歌是放在CD上，CD需要播放机才能转成实际发出的声音

另外还需5个步骤：

① 取得Sequencer并将它打开。

```
Sequencer player = MidiSystem.getSequencer();
player.open();
```

② 创建新的Sequence。

```
Sequence seq = new Sequence(timing,4);
```

③ 从Sequence中创建新的Track。

```
Track t = seq.createTrack();
```

④ 填入MidiEvent并让Sequencer播放。

```
t.add(myMidiEvent1);
player.setSequence(seq);
```



史上第一个声音播放程序

把它输入然后运行看看。你会听到某物发出单一钢琴音。这是你个人的一小步，人类史上无关痛痒的一大步。

```
import javax.sound.midi.*;
```

别忘了要import进midi的包

```
public class MiniMiniMusicApp {

    public static void main(String[] args) {
        MiniMiniMusicApp mini = new MiniMiniMusicApp();
        mini.play();
    } // 关闭main

    public void play() {

      try {
```

❶

```
            Sequencer player = MidiSystem.getSequencer();
            player.open();
```

取得Sequencer并将其打开

❷

```
            Sequence seq = new Sequence(Sequence.PPQ, 4);
```

先不用管参数的意义

❸

```
            Track track = seq.createTrack();
```

要求取得Track

❹



```
            ShortMessage a = new ShortMessage();
            a.setMessage(144, 1, 44, 100);
            MidiEvent noteOn = new MidiEvent(a, 1);
            track.add(noteOn);

            ShortMessage b = new ShortMessage();
            b.setMessage(128, 1, 44, 100);
            MidiEvent noteOff = new MidiEvent(b, 16);
            track.add(noteOff);
```

对Track加入几个MidiEvent，要注意的是setMessage()的参数，以及MidiEvent的constructor，下一页会讨论

```
            player.setSequence(seq);
```

将Sequence送到Sequencer上

```
            player.start();
```

开始播放

```
      } catch (Exception ex) {
            ex.printStackTrace();
      }
   } // 关闭播放
} // 关闭类
```

制作 MidiEvent（乐曲信息）

MidiEvent是组合乐曲的指令。一连串的MidiEvent就好像是乐谱一样。我们会在乎的MidiEvent大部分都与描述要做的事情以及时机有关。因为MidiEvent是非常琐碎的描述，所以你必须指定何时开始播放某个音符（NOTE ON事件）以及何时停止（NOTE OFF事件），因此你可以想象在“开始发出 G 音”之前发出“停止播放G音”是没有作用的。

MIDI指令实际上会放在Message对象中，MidiEvent是由Message加上发音时机所组成的。也就是说Message会带有“开始播放C”指令，并伴随着“于第四拍执行指令”的信息。

因此我们会同时需要MidiEvent与Message。

Message描述做什么，而MidiEvent指定何时做。

MidiEvent用来指示在何时执行什么操作。

每个指令都必须包括该指令的执行时机。

也就是说，乐声应该在哪一拍发响。

① 创建Message。

```
ShortMessage a = new ShortMessage();
```

② 置入指令。

```
a.setMessage(144, 1, 44, 100);
```

这代表发出44音符，其余的参数下一页会说明

③ 用Message创建 MidiEvent。

```
MidiEvent noteOn = new MidiEvent(a, 1);
```

在第一拍启动a这个Message

④ 将MidiEvent加到Track中

```
track.add(noteOn);
```

Track带有全部的MidiEvent对象，Sequence会根据事件时间组织它们，然后Sequencer会根据此顺序来播放，同一时间可以执行多个操作，例如和弦声音或不同乐器的声音

MIDI的Message是MidiEvent的关键

MIDI的Message带有事件中要执行什么操作的部分，也就是要sequencer实际执行的指令。指令的第一个参数是信息的类型，后面的3个参数要看信息的类型来决定它们的意义。例如144 类型的信息代表“NOTE ON”。为了要带出NOTE ON指令，sequencer还需要知道几件事。你可以想象sequencer会问到：“好啊，我会发这个音，但是要用哪个频道？是鼓声的还是钢琴的频道？音量要多大？

要创建MIDI的Message，用ShortMessage的实例调用setMessage()，传入该信息的4个参数。但要记住，你还需要把信息加上执行时机装入事件中。

信息的格式

第一个参数是信息类型，其余3个要看信息类型而定。

Message 是执行的内容， MidiEvent 是执行的时机

类型　频道　音符　音道

```
a.setMessage(144,  1,  44,  100);
```

这是一个NOTE ON信息，因此其余三项参数用来描述所要发出的声响

① 信息类型。

144代表打开



128代表关闭



② 频道。

每个频道代表不同的演奏者。例如一号频道是吉他、二号是Bass；或者可以像Iron Maiden用3把不同音色的吉他编制。

③ 要发出的音符。

从0～127代表不同音高。



④ 音道。

用多大的音道按下？0几乎听不到，100算是差不多。

改变信息

现在你已经知道Midi信息的内容，可以开始进行实验。你可以改变播放的音符、音长、加入更多音符、甚或是改变乐器。

① 改变音符。

试试看用0 ~ 127之间的数字来改变。

```
a.setMessage(144, 1, 20, 100);
```



② 改变音长。

对NOTE OFF的事件作些音长的变化。

```
b.setMessage(128, 1, 44, 100);
MidiEvent noteOff = new MidiEvent(b, 3);
```



③ 改变乐器。

在播放信息之前加入新的信息，换个不一样的乐器，从0~127之间找，看看还有什么音色。

```
first.setMessage(192, 1, 102, 0);
```

改变乐器的信息

用第一个频道

换成102的乐器



第二版：使用命令列参数

这个版本还是播放单一的音符而已，但你可以使用命令列参数来改变乐器和音符。试试看传入两个介于0 ～127之间的整数参数。第一个参数设定乐器，第二个整数设定音符。

```
import javax.sound.midi.*;

public class MiniMusicCmdLine {   // 这是第一个

   public static void main(String[] args) {
        MiniMusicCmdLine mini = new MiniMusicCmdLine();
        if (args.length < 2) {
            System.out.println("Don't forget the instrument and note args");
        } else {
            int instrument = Integer.parseInt(args[0]);
            int note = Integer.parseInt(args[1]);
            mini.play(instrument, note);
        }
   } // 关闭main

   public void play(int instrument, int note) {

      try {

         Sequencer player = MidiSystem.getSequencer();
         player.open();
         Sequence seq = new Sequence(Sequence.PPQ, 4);
         Track track = seq.createTrack();

         MidiEvent event = null;

         ShortMessage first = new ShortMessage();
         first.setMessage(192, 1, instrument, 0);
         MidiEvent changeInstrument = new MidiEvent(first, 1);
         track.add(changeInstrument);

         ShortMessage a = new ShortMessage();
         a.setMessage(144, 1, note, 100);
         MidiEvent noteOn = new MidiEvent(a, 1);
         track.add(noteOn);

         ShortMessage b = new ShortMessage();
         b.setMessage(128, 1, note, 100);
         MidiEvent noteOff = new MidiEvent(b, 16);
         track.add(noteOff);
         player.setSequence(seq);
         player.start();

      } catch (Exception ex) {ex.printStackTrace();}
   } // 关闭播放
} // 关闭类
```

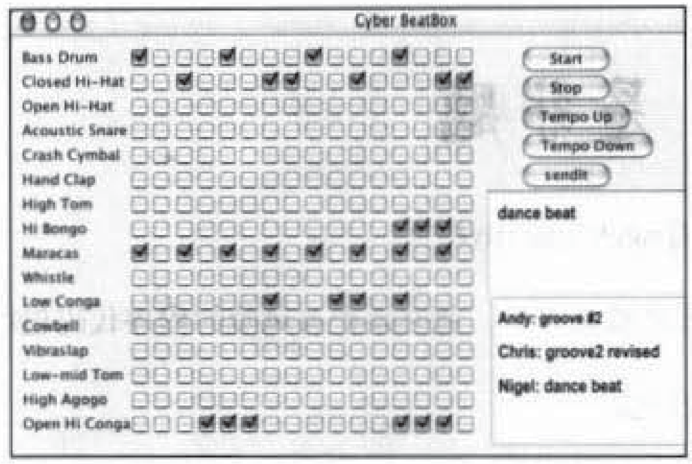
以两个介于0～127的整数参数来运行

```
File Edit Window Help Attenuate
%java MiniMusicCmdLine 102 30
%java MiniMusicCmdLine 80 20
%java MiniMusicCmdLine 40 70
```

接下来的章节会继续料理程序

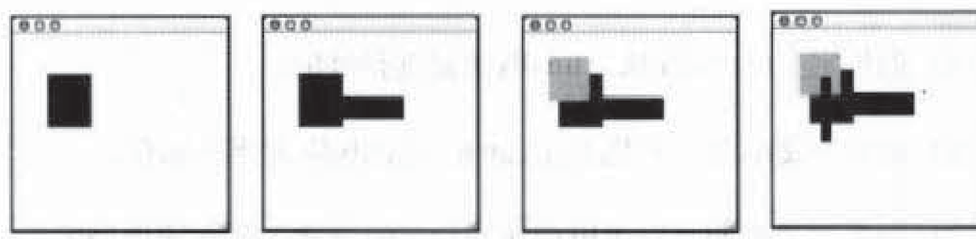
第15章：目的地

完成之后我们就会有一个BeatBox程序，它同时也是Drum Chat的客户端程序。我们需要学习GUI、I/O、网络和线程等。接下来的3章就包括这些内容。



第12章：MIDI事件

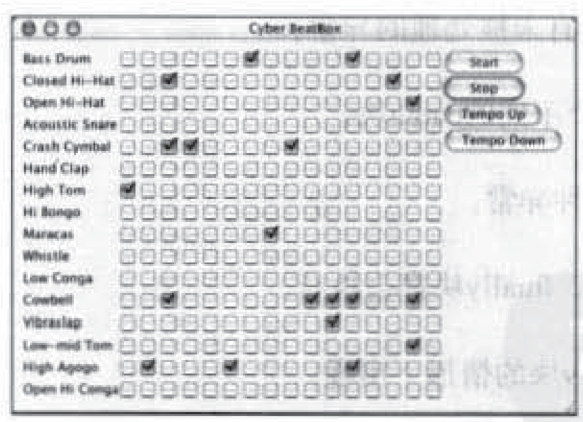
这一章会创建出一个小小的“MTV”，它会根据MIDI音乐的节奏来画出随机的字符。这一章的内容能让我们学习到MIDI事件的处理。



beat one　beat two　beat three　beat four……

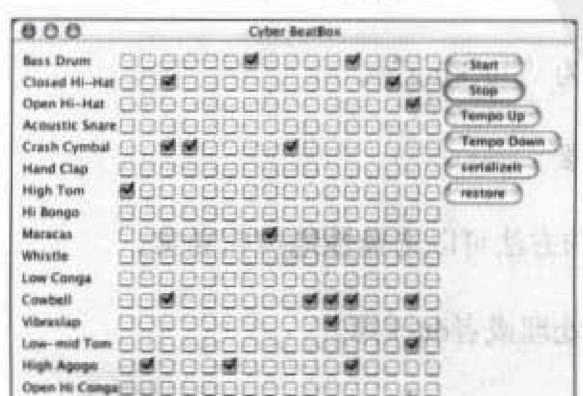
第13章：独立的BeatBox

我们确实创建出BeatBox、GUI等功能。但这还是很受限的版本。没有存储和还原的功能，也无法通过网络通信。



第14章：存储和还原

现在可以存盘与加载还原以再度播放了。这让我们可以准备好完成最终版本，把存盘发到网络另一端的chat服务器。





这一章讨论Java的异常。你的任务是辨别下面的陈述哪些是对的？哪些是错的？

是非题

（1）try块必须要跟着catch与finally块后。

（2）如果遇到编译器的检查异常，就必须把有风险的程序代码包在try/catch块中。

（3）catch块可以多态化。

（4）只有编译器的检查异常才会被捕获。

（5）如果定义try/catch块，finally块是选择性的。

（6）如果定义try块，可以加上catch、finally块或两者都有。

（7）如果方法声明可以抛出编译器检查的异常，则必须把抛出异常的程序代码包在try/catch块中。

（8）main()方法必须处理所有未被处理的异常。

（9）单一的try块可以有多个不同的catch块。

（10）一个方法只能抛出一种异常。

（11）不管有没有抛出异常，finally块都会执行。

（12）finally块可以在没有try块的情形下出现。

（13）try块可以在没有catch或finally块的情形下单独出现。

（14）异常的处理有时被称为“ducking”。

（15）catch块的顺序并不重要。

（16）带有try块和finally块的方法可以选择性地声明异常。

（17）运行期的异常必须要处理或者duck掉。





排排看

下面是被打散的Java程序片段。你是否能够将它们重新排列以成为可以编译与执行并产生如同下方的输出结果？注意到有些括号已经遗失，所以你可以在认为有需要时自行补上。

```
System.out.print( r );
```

```
try {
```

```
System.out.print( t );
```

```
doRisky(test);
```

```
System.out.println( s );
```

```
} finally {
```

```
System.out.print( o );
```

```
class MyEx extends Exception { }

public class ExTestDrive {
```

```
System.out.print( w );
```

```
if ( yes .equals(t)) {
```

```
System.out.print( a );
```

```
throw new MyEx();
```

```
} catch (MyEx e) {
```

```
static void doRisky(String t) throws MyEx {
  System.out.print( h );
```

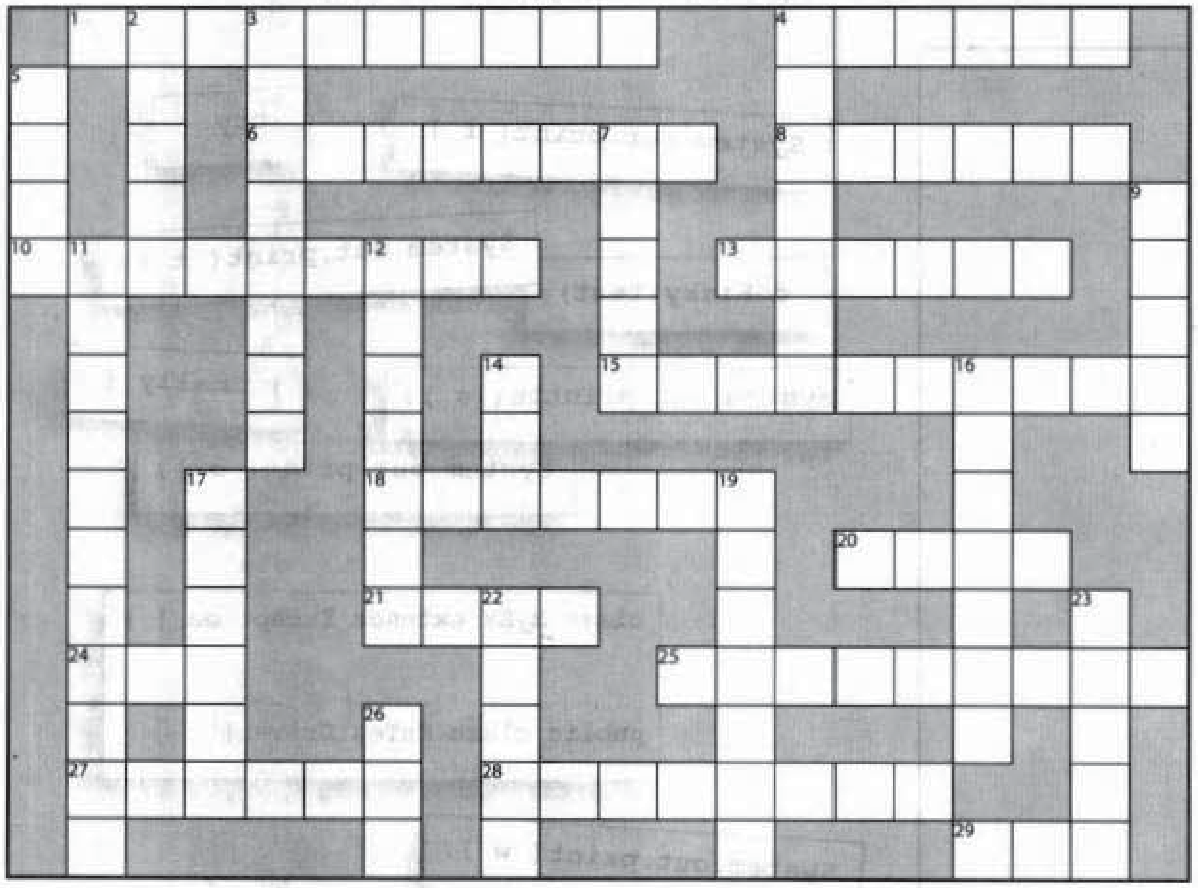
```
public static void main(String [] args) {
  String test = args[0];
```

File Edit Window Help ThrowUp

```
% java ExTestDrive yes
thaws

% java ExTestDrive no
throws
```

JavaCross 7.0



你知道该怎么做了吧？

(当然不是直接偷看答案)

横排提示：

1. To give value
4. Flew off the top
6. All this and more!
8. Start
10. The family tree
13. No ducking
15. Problem objects
18. One of Java's '49'
20. Class hierarchy
21. Too hot to handle
24. Common primitive
25. Code recipe
27. Unruly method action
28. No Picasso here
29. Start a chain of events

竖排提示：

2. Currently usable
3. Template's creation
4. Don't show the kids
5. Mostly static API class
7. Not about behavior
9. The template
11. Roll another one off the line
12. Javac saw it coming
14. Attempt risk
16. Automatic acquisition
17. Changing method
19. Announce a duck
22. Deal with it
23. Create bad news
26. One of my roles

补充提示：

横排提示
6. A Java child
8. Start a method
13. Instead of declare
20. Also a type of collection
21. Quack
27. Starts a problem
28. Not Abstract

竖排提示
2. Or a mouthwash
3. For ____ (not example)
5. Numbers...
9. Only public or default
16. ____ the family fortune
17. Not a 'getter'

(字谜的问题部分保留原汁原味的英文，请自己动手查字典！)

练习解答

是非题

(1) 错，任一或两者皆可。

(2) 错，也可以声明异常。

(3) 对。

(4) 错，运行期间异常也会被捕获到。

(5) 对。

(6) 对。

(7) 错，只要声明就够了。

(8) 错，但如果连它也不管，那Java虚拟机只好中断。

(9) 对。

(10) 错。

(11) 对。

(12) 错。

(13) 错。

(14) 错，duck在这里有声明的意思。

(15) 错，越广泛的异常越要在后面。

(16) 错，如果没有catch就要duck。

(17) 错。

排排看

```
class MyEx extends Exception { }

public class ExTestDrive {

  public static void main(String [] args) {
    String test = args[0];
    try {

      System.out.print("t");

      doRisky(test);

      System.out.print("o");

    } catch ( MyEx e) {

       System.out.print("a");

    } finally {

       System.out.print("w");
    }
    System.out.println("s");
  }

  static void doRisky(String t) throws MyEx {
    System.out.print("h");

    if ("yes".equals(t)) {

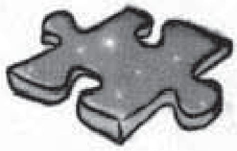
      throw new MyEx();
    }

    System.out.print("r");

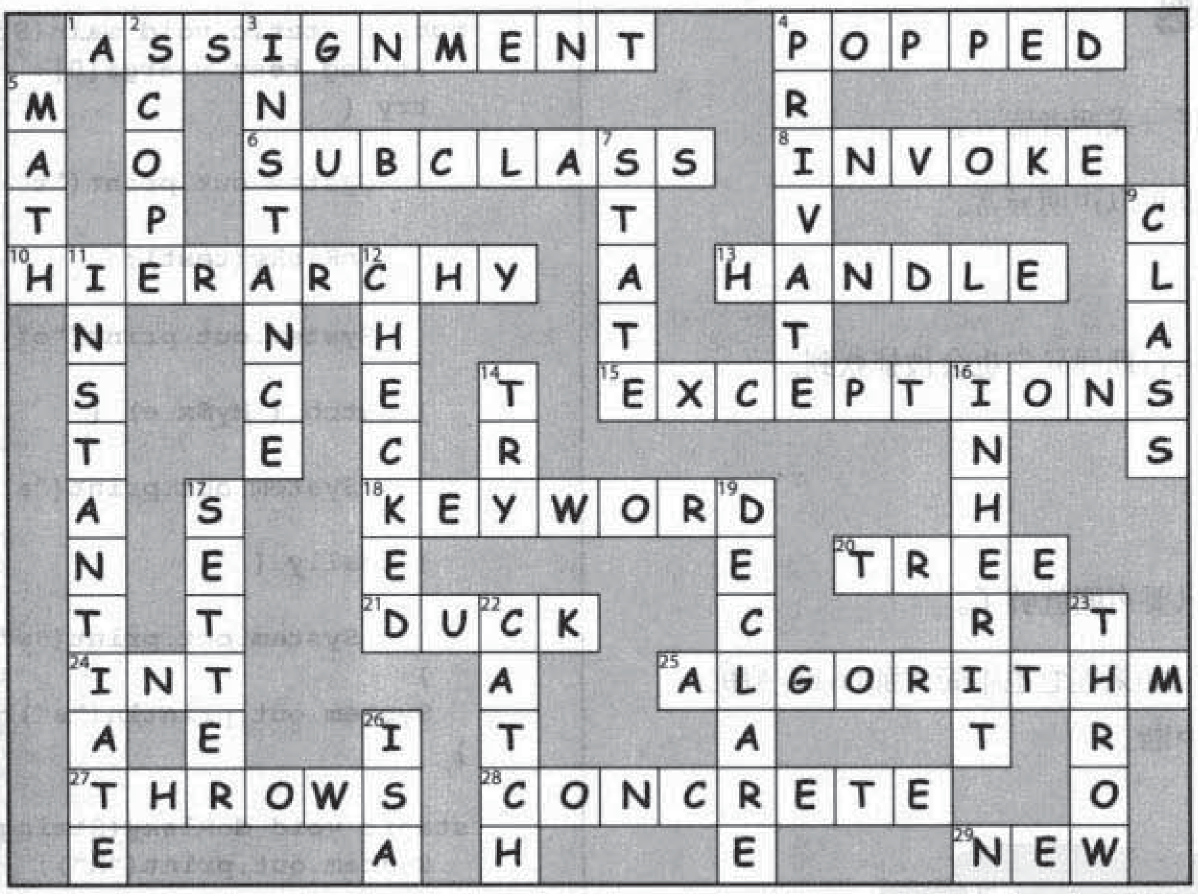
  }
}
```

```
File Edit Window Help Chill
% java ExTestDrive yes
thaws

% java ExTestDrive no
throws
```



JavaCross答案



12 图形用户接口

看图说故事



面对现实吧，你得做出图形用户接口。 如果你要创建别人会使用的应用程序，就必须要用到图形接口。如果你是写给自己用，那应该也会想要有个图形接口吧。即使你只会写服务器端的应用程序，客户端的用户接口是由Web页面组成的，早晚你还是得写图形化的工具接口。当然啦，命令行的程序一样可以使用，但逃避并不是个好办法。命令行又差又没适应性，而且也不好用。我们会用两章的篇幅来讨论GUI，并且同时学习包括事件处理与内部类别等Java语言的关键功能。在这一章我们会制作按钮、绘制屏幕画面、显示jpeg图文件，另外还做动画。

一切都从window开始

JFrame是个代表屏幕上window的对象。你可以把button、checkbox、text字段等接口放在window上面。标准的menu也可以加到上面，并且能够带最小化、最大化、关闭等图标。

JFrame的长相会依据所处的平台而有所不同。下图是JFrame在Mac OS X上的样子：



"如果再让我看到一个命令行的程序，你就得走人"



这是一个带有menu与button以及radio button的JFrame

将组件加到window上

一旦创建出JFrame之后，你就可以把组件（widget）加到上面。有很多的Swing组件可以使用，它们是在javax.swing这个包中。最常使用的组件包括：JButton、JRadioButton、JCheckBox、JLabel、JList、JScrollPane、JSlider、JTextArea、JTextField和JTable等。大部分都很容易使用，但像JTable这些是有点复杂的。

创建GUI真简单：

① 创建frame。

```
JFrame frame = new JFrame();
```

② 创建widget。

```
JButton button = new JButton("click me");
```

③ 把widget加到frame上。

```
frame.getContentPane().add(button);
```

组件不会直接加到frame上，你可以把frame想象成window的框，组件是加到window的pane上面

④ 显示出来。

```
frame.setSize(300,300);
frame.setVisible(true);
```

你的第一个GUI

```
import javax.swing.*;

public class SimpleGui1 {
    public static void main (String[] args) {

        JFrame frame = new JFrame();
        JButton button = new JButton("click me");

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.getContentPane().add(button);

        frame.setSize(300,300);

        frame.setVisible(true);
    }
}
```

别忘了import进这个包

创建frame和button

这一行程序会在window关闭时把程序结束掉

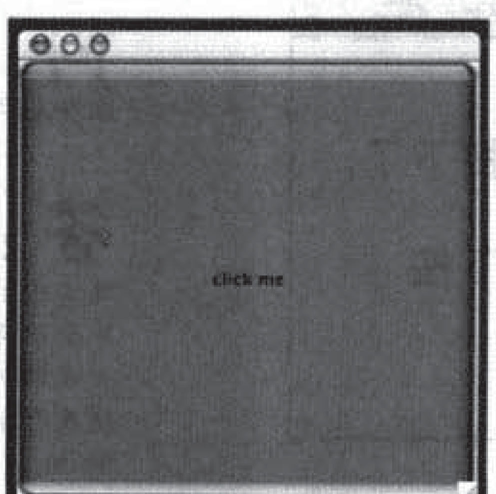
把button加到frame的pane上

设定frame的大小

最后把frame显示出来！

执行时会是怎样的情景：

%java SimpleGui1



啊！

好大一个按钮啊！

这个按钮会填满整个frame。稍后我们会讨论如何控制按钮的大小和位置。

PDG

但是按钮没有功能……

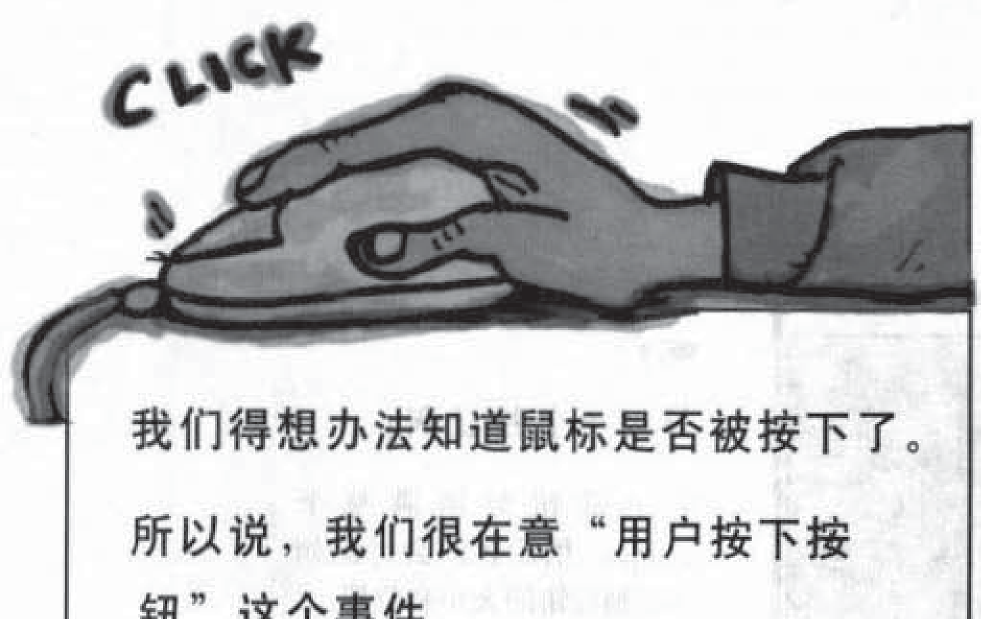
其实并不是这样。当你按下按钮时，它会显示出“被按下”或“被推倒”的外观（实际的样子要看平台，但一定会显示出让你能够分辨的形状）。

所以这个问题应该要这样问：如何让按钮在用户按下时执行特定的工作？

需要这两项：

1. 被按下时要执行的方法（也就是按钮的任务）。

2. 检测按钮被按下的方法，换句话说，就是按钮的感应装置。



我们得想办法知道鼠标是否被按下了。

所以说，我们很在意“用户按下按钮”这个事件。

* pushed的按钮应该要翻译成维持在按下状态的按钮，或者你可以把它想象成开关的状态，翻译成被推倒是因为译者个人的兴趣……

there are no Dumb Questions

问： 按钮在Windows上面运行的时候会不会就长得跟其他Windows程序的按钮一样？

答： 想要的话也可以。你能够使用一些核心函数库的“look and feel”类型，该类来控制界面的外观和操作感受。大部分情况下都有两种可以选：称为Metal的Java标准和平台原始界面两种。本书展示的是Mac OS X的Aqua外观或Metal外观。

问： 能不能在Windows上使用Aqua外观？

答： 你做梦。不是所有外观和操作感受都能够在不同平台上发现的。要取得跨平台的相同外观就得使用Metal，不然就不要指定，让外观使用平台的默认值。

问： 听说Swing很慢并且根本没人用？

答： 以前是这样，现在可就不同了。如果机器很烂，那你可能会用得很痛苦，但对于稍微像样一点的机器来说，你根本感受不到速度有差别。很多应用程序都用到Swing。

取得用户的事件

假设你想要把按钮上面的文字在用户按下按钮时从“click me”变成“I've been clicked”。首先你得编写改变按钮文字的方法（查一下API你就会知道怎么改）：

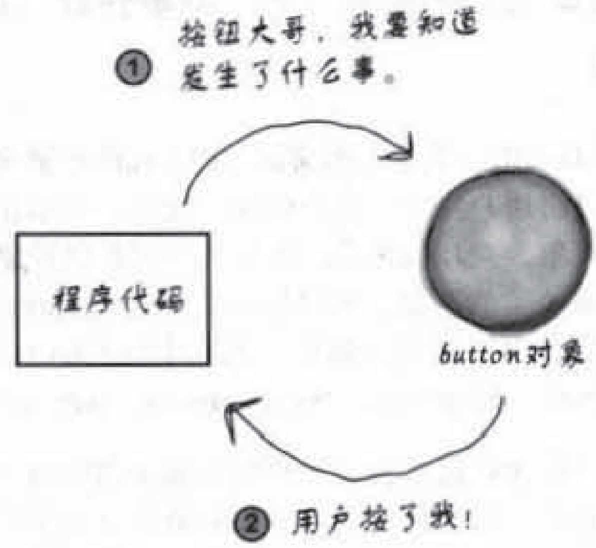
```
public void changeIt() {
    button.setText("I've been clicked!");
}
```

然后呢？这个方法应该在什么时候执行？我们怎么知道按钮被按下去呢？

在Java上，取得与处理用户操作事件的过程称为even-handling。Java有许多不同的事件类型，大多数都与GUI上的用户操作有关。如果用户按下了按钮，就会产生事件。这是一个关于用户想要采取启动按钮功能的事件。如果这是个“放慢节奏”的按钮，就表示说用户想要让节奏慢一点。所以最直截了当的事件就是这种表明要执行某种操作的事件。

对这样的按钮来说，你不会在乎像是按钮正被按住或按钮已经放开这类的过渡性事件。你只想知道用户是不是想要采取某种行动。也就是说你不管鼠标是否一直按着不放之类的事情，只管用户真正的意图！

首先，按钮要知道它的作用。



其次，按钮要在按键事件发生时调用执行功能的方法。

BRAIN POWER

（1）你要怎样告诉按钮对象你在乎它的事件？如何表白你对它的关心？

（2）假设你不能把方法的名字告诉按钮，按钮要怎么通知你？我们怎样确保当某特定事件发生时会调用指定的方法？

如果想要知道按钮的事件，就会监听事件的接口

监听接口是介于监听（你）与事件源（按钮）间的桥梁

实现监听接口让按钮有一个回头调用程序的方式。interface正是声明调用（call-back）方法的地方。

Swing的GUI组件是事件的来源。以Java的术语来说，事件来源是个可以将用户操作（点击鼠标、按键、关闭窗口等）转换成事件的对象。对Java而言，事件几乎都是以对象来表示。它会是某种事件类的对象。如果你查询API中的java.awt.event这个包，就会看到一组事件的类（名称中都有Event）。你会看到MouseEvent、KeyEvent、WindowEvent、ActionEvent等等。

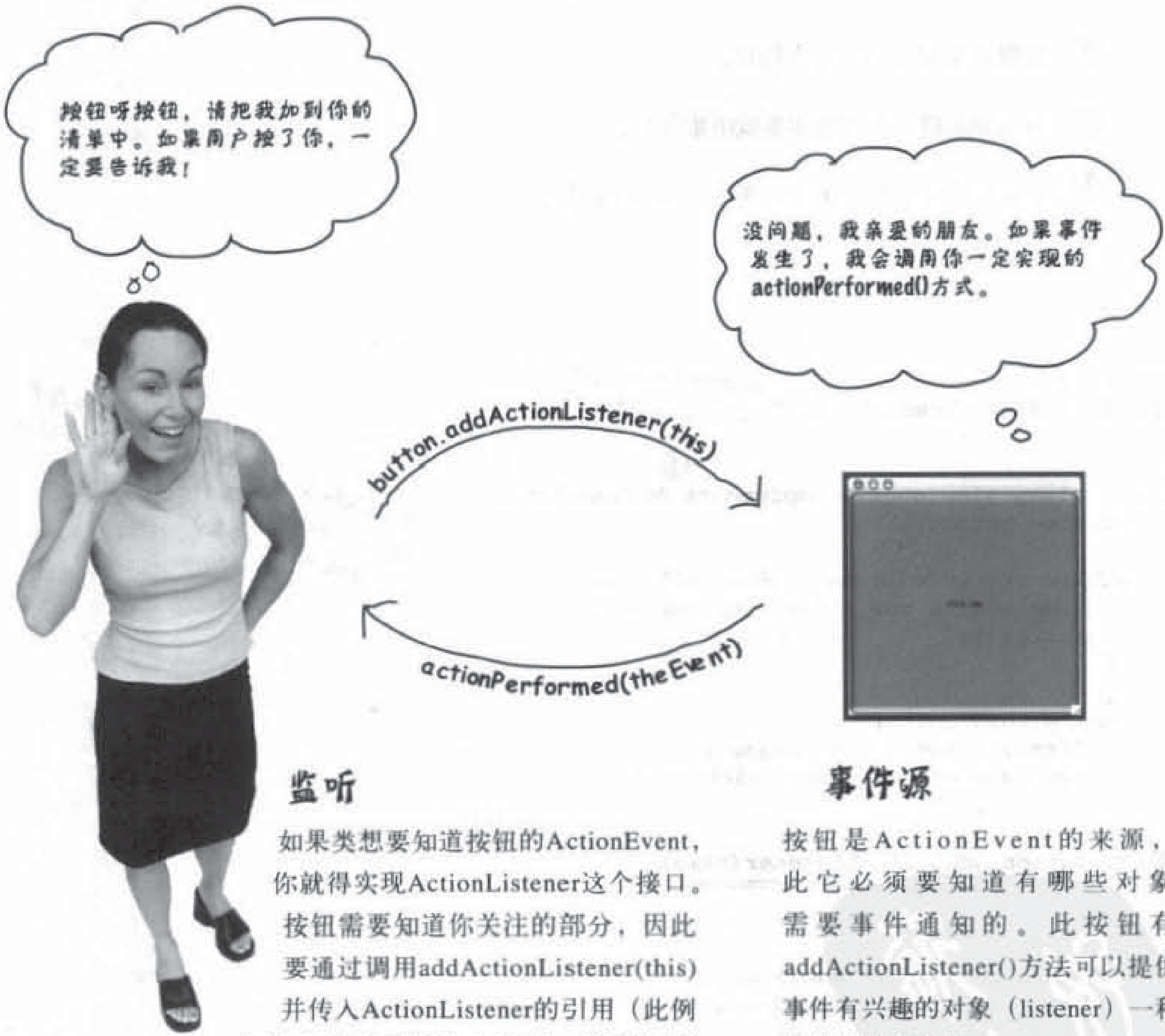
事件源（例如按钮）会在用户做出相关动作时（按下按钮）产生事件对象。你的程序在大多数的情况下是事件的接受方而不是创建方。也就是说，你会花较多的时间当监听者而不是事件来源。

每个事件类型都有相对应的监听者接口。想要接收MouseEvent的话就实现MouseListener这个接口。想要WindowEvent吗？实现WindowListener。记得接口的规则：要实现接口就得声明这件事（各位乡亲注意啦，Dog实现Pet了！），这代表你必须把接口中所有的方法都实现出来。

某些接口不只有一个方法，因为事件本身就有不同的形态。以MouseListener为例，事件就有mousePressed、mouseReleased、MouseMoved等。虽然都是MouseEvent，每个鼠标事件都在接口中有不同的方法。如果有实现MouseListener的话，mousePressed()就会在用户按下鼠标的时候被调用。当按键放开时会调用mouseReleased()。因此鼠标事件只有MouseEvent一种事件对象，却有不同的事件方法来表示不同类型的鼠标事件。



监听和事件源如何沟通



监听

如果类想要知道按钮的ActionEvent，你就得实现ActionListener这个接口。按钮需要知道你关注的部分，因此要通过调用addActionListener(this)并传入ActionListener的引用（此例中就是你自己的这个程序，所以使用this）来向按钮注册。按钮会在该事件发生时调用该接口上的方法。而作为一个ActionListener，编译器会确保你实现此接口的actionPerformed()。

事件源

按钮是ActionEvent的来源，因此它必须要知道有哪些对象是需要事件通知的。此按钮有个addActionListener()方法可以提供对事件有兴趣的对象（listener）一种表达此兴趣的方法。

当按钮的addActionListener()方法被调用时（因为某个listener的调用），它的参数会被按钮存到清单中。当用户按下按钮时，按钮会通过调用清单上每个监听的actionPerformed()来启动事件。

取得按钮的ActionEvent

- 实现ActionListener这个接口。
- 向按钮注册（告诉它你要监听事件）。
- 定义事件处理的方法（实现接口上的方法）。

```
import javax.swing.*;
import java.awt.event.*;   ← import进ActionListener和ActionEvent所在的包

public class SimpleGui1B implements ActionListener {   ← 实现此接口。这表示SimpleGui1B是个ActionListener（事件只会通知有实现ActionListener的类）
    JButton button;

    public static void main (String[] args) {
       SimpleGui1B gui = new SimpleGui1B();
       gui.go();
    }

    public void go() {
       JFrame frame = new JFrame();
       button = new JButton("click me");

       button.addActionListener(this);   ← 向按钮注册

       frame.getContentPane().add(button);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(300,300);
       frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {   ← 实现interface上的方法……这是真正处理事件的方法！
       button.setText("I've been clicked!");
    }
}
```

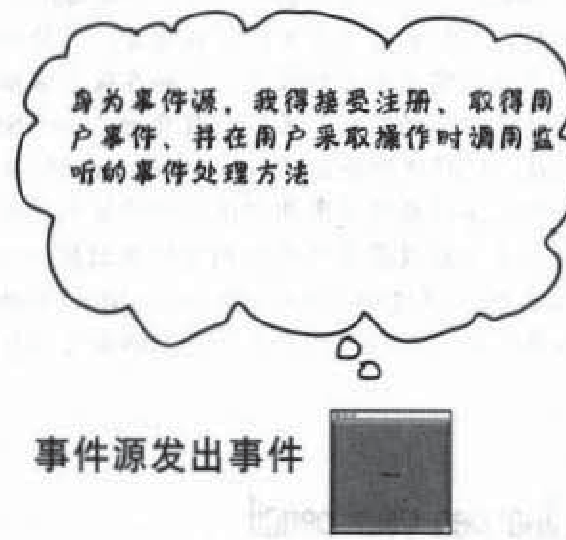
按钮会以ActionEvent对象作为参数来调用此方法

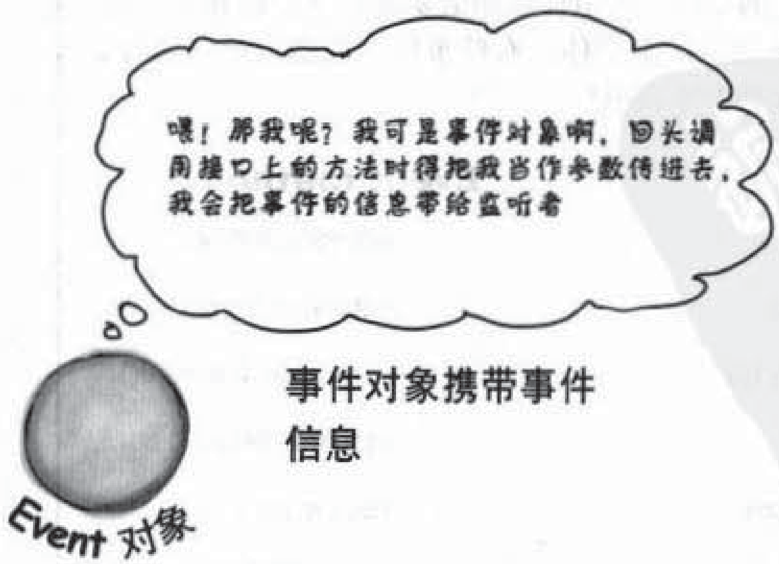
监听、事件源与事件

对一般的Java程序员来说，职业生涯中很少会有成为事件源的机会（不管你有多爱表现、多喜欢上台表演、多希望成为注目的焦点，就算经常把照片贴上网络也一样）。

接受事实吧，你的工作是监听（如果能够做好这件事，你也会成为异性朋友眼中诚恳的“好人”！）。







there are no Dumb Questions

问： **我为什么不是事件源？**

答： 你也可以是。我们只是说大部分的时间你会是接收事件的一方而不是来源（至少在你早期的Java职业生涯中不是）。大部分你会用到的事件是由Java API中的类发出的，而你会是这些事件的监听者。无论如何，你可以设计需要自定义事件的程序，例如在股票涨幅超过一定程度的时候从你的股票交易监控程序抛出一个StaockMarketEvent。此例中你会创建出StockWatcher对象来当作事件源，并对你的自定义事件创建监听的接口，提供注册的方法等。之后在股票事件发生的时候就创建一个StockEvent对象并把它通过调用stockChanged(StockEvent)的方法来传给监听者。别忘记每个事件类型都要有相对应的监听接口。

问： **我看不出传给事件调用方法的事件对象有什么重要性。调用mousePressed时可能有哪些信息会被用到？**

答： 大部分情况下你不会用到事件对象。它只不过是个携带事件数据的载体。但有时你也会需要查询事件的特定细节。例如你的mousePressed()被调用时，你知道有鼠标的按钮被按下。但如果你想要知道鼠标的坐标呢？

或者有时候你会想要对相同的监听注册多个对象。举例来说，计算器程序会有10个按键，且因为都做相同的事情，所以你可能不想为每个按键个别地制作监听。所以当你收到事件时，可以设计成用事件对象的信息来判别哪一个按键钮发了事件。

Sharpen your pencil

下列的每个组件（widget，用户接口对象）是一或多个事件的来源。把组件可能会发出的事件连起来。有些组件会有多个事件，有些事件可能会有多个来源。看不懂就查字典吧！

Widgets	Event methods
check box	windowClosing()
text field	actionPerformed()
scrolling list	itemStateChanged()
button	mousePressed()
dialog box	keyTyped()
radio button	mouseExited()
menu item	focusGained()

你怎么知道某个对象是否为事件的来源呢？

查询API文件。

好的，查什么？

以“add”开头“Listener”且取用listener接口参数的方法！像是：

addKeyListener(KeyListener k)

有这种method的class就是KeyEvent的来源。

回到图形上面……

我们已经大概知道事件的运行方式（后面还有更详细的说明），现在回到在屏幕的绘制上。在回到事件处理前，先来花一点点时间来玩一下图形。

在GUI上面加东西的3种方法：

① **在frame上放置widget**

加上按钮、窗体、radio button等。

```
frame.getContentPane().add(myButton);
```

javax.swing这个包上面有超过一打的widget类型。

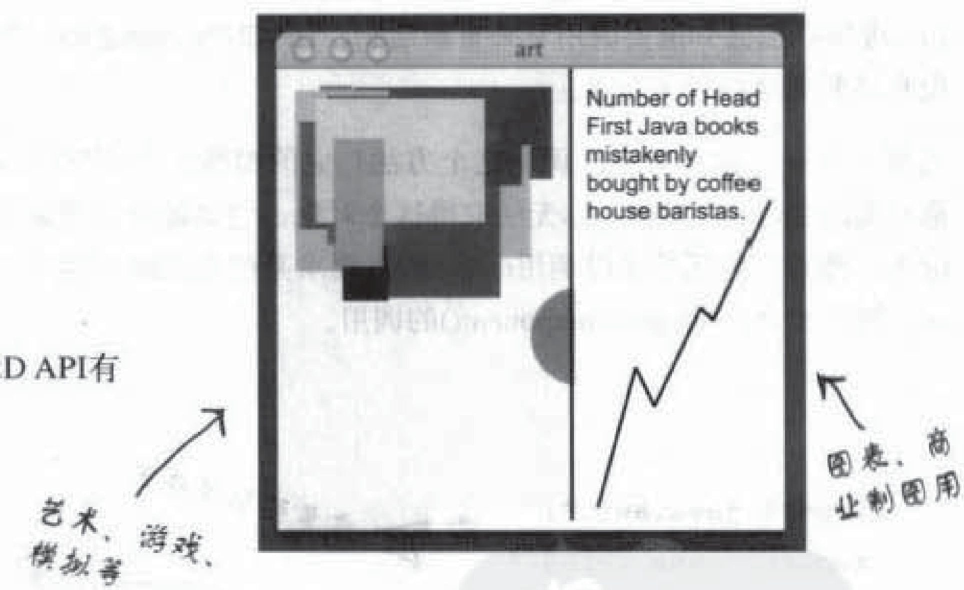


② **在widget上绘制2D图形**

使用graphics对象来绘制图形。

```
graphics.fillOval(70, 70, 100, 100);
```

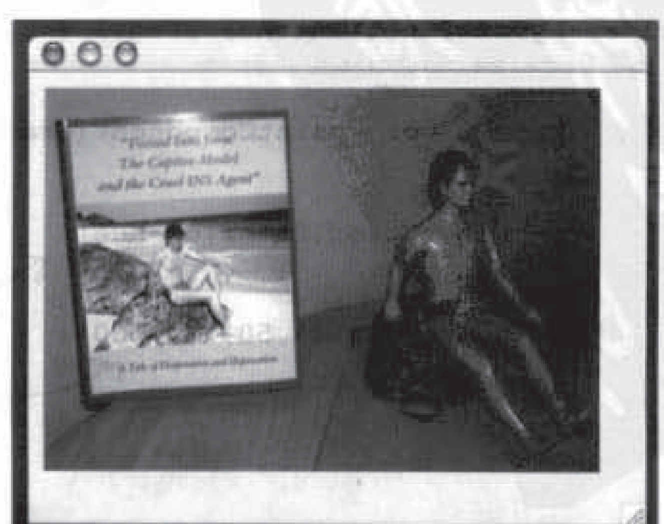
你可以画上很多的方块和圆圈，Java2D API有很多好玩、复杂的图形方法。



③ **在widget上绘制JPEG图**

把图形画在widget上。

```
graphics.drawImage(myPic,10,10,this);
```



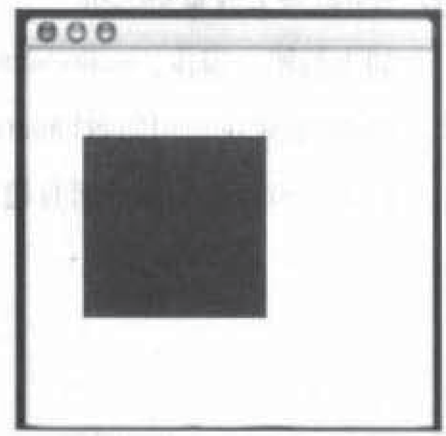
自己创建的绘图组件

如果你要在屏幕放上自己的图形，最好的方式是自己创建出有绘图功能的widget。你把widget放在frame上，如同按钮或其他的widget一样，不同之处在于它会按照你所要的方式绘制。你还可以让图形移动、表现动画效果或在点选的时候改变颜色。

简单得不得了。

创建JPanel的子类并覆盖掉paintComponent()这个方法。

所有绘图程序代码都在paintComponent()里面。把这个方法想象成会被系统告知要把自己画出来的方法。如果你要画的是圆圈，就写画圆圈的程序。当你的panel所处的frame显示的时候，paintComponent()就会被调用。如果用户缩小window或选择最小化，Java虚拟机也会知道要调用它来重新绘制。任何时候Java虚拟机发现有必要重绘都会这么做。



还有一件事，你不会自己调用这个方法！它的参数是个跟实际屏幕有关的Graphics对象，你无法取得这个对象，它必须由系统来交给你。然而，你还是可以调用reapint()来要求系统重新绘制显示装置，然后才会产生paintComponent()的调用。

```
import java.awt.*;
import javax.swing.*;

class MyDrawPanel extends JPanel {

    public void paintComponent(Graphics g) {

        g.setColor(Color.orange);

        g.fillRect(20,50,100,100);
    }
}
```

需要引用这些包

创建JPanel的子类

这是非常重要的方法，你决不能自己调用，要由系统来调用

你可以把g想象成绘图装置，告诉它要用什么颜色画出什么形状

在paintComponent()中可以做的事情

来看一下其他可以在paintComponent()中做的事情，其中最有趣的莫过于自己做些试验。试试看操弄一下数字，并查询Graphics这个 类的API（稍后我们会看到更多的说明）。

显示JPEG

```
public void paintComponent(Graphics g) {

    Image image = new ImageIcon("catzilla.jpg").getImage();

    g.drawImage(image,3,4,this);

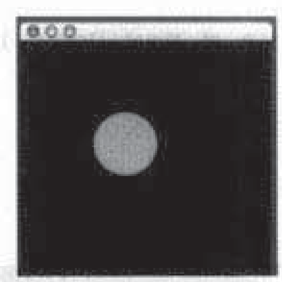
}
```

图文件名

这个坐标代表图案左上角的位置离panel的左方边缘3个像素，离顶端向下4个像素



在黑色背景画上随机色彩的圆圈



```
public void paintComponent(Graphics g) {

   g.fillRect(0,0,this.getWidth(), this.getHeight());

   int red = (int) (Math.random() * 255);
   int green = (int) (Math.random() * 255);
   int blue = (int) (Math.random() * 255);

   Color randomColor = new Color(red, green, blue);
   g.setColor(randomColor);
   g.fillOval(70,70,100,100);
}
```

以默认颜色填充

前两个参数是起点的坐标，后面两个参数分别是宽度和高度，此处取得本身的宽高，因此会把panel填满

传入3个参数来代表RGB值

填满参数指定的椭圆型区域

在每个Grahphics引用的后面都有个Grahpics2D对象

paintComponent()的参数被声明为Graphics类型（java.awt.Grahpics）。

```
public void paintComponent(Graphics g) { }
```

因此参数g是个Graphics对象。这代表它可能是个Graphics的子类（因为多态的缘故），事实上就是这样。

由g参数所引用的对象实际上是个Grahpics2D的实例。

为何要知道？因为有些在Graphics2D引用上可以做的事情不能在Grahpics引用上做。Graphics2D对象可以做的事情比Graphics对象更多，实际上躲在Graphics引用的后面是个Graphics2D对象。

记得多态的问题，编译器会根据引用的类型而不是实际对象来判定你能够调用哪些方法。如果你有个Dog对象是由Animal引用变量来引用方法的：

```
Animal a = new Dog( );
```

那你就不能让a吠：

```
a.bark( );
```

就算你明知道那是个Dog也一样。但你还是可以把它转回成Dog：

```
Dog d = (Dog) a;
d.bark( );
```

因此Graphics对象的底限是这样的：

如果你要调用Graphics2D类的方法，就不能直接使用g参数。但你可以将它转换成Graphics2D变量。

```
Graphics2D g2d = (Graphics2D) g;
```

可以对Graphics引用调用的 方法：

drawImage()

drawLine()

drawPolygon

drawRect()

drawOval()

fillRect()

fillRoundRect()

setColor()

转换成Graphics2D对象：

```
Graphics2D g2d = (Graphics2D) g;
```

可以对Graphics2D引用调用的方法：

fill3DRect()

draw3DRect()

rotate()

scale()

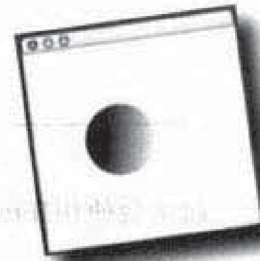
shear()

transform()

setRenderingHints()

（这不是完整的列表，查阅API文件可以取得最完整的说明）

纯色算不了什么，还有渐层颜色可以做出很棒（或很俗）的效果



实际上是个Graphics2D对象

```
public void paintComponent(Graphics g) {

    Graphics2D g2d = (Graphics2D) g;

    GradientPaint gradient = new GradientPaint(70,70,Color.blue, 150,150, Color.orange);

    g2d.setPaint(gradient);

    g2d.fillOval(70,70,100,100);

}
```

将它的类型转换成Graphics2D

起点　开始的颜色　终点　渐层最后的颜色

将虚拟的"笔刷"换成渐层

用目前的笔刷设定来填满椭圆型的区域

```
public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    int red = (int) (Math.random() * 255);
    int green = (int) (Math.random() * 255);
    int blue = (int) (Math.random() * 255);
    Color startColor = new Color(red, green, blue);

    red = (int) (Math.random() * 255);
    green = (int) (Math.random() * 255);
    blue = (int) (Math.random() * 255);
    Color endColor = new Color(red, green, blue);

    GradientPaint gradient = new GradientPaint(70,70,startColor, 150,150, endColor);
    g2d.setPaint(gradient);
    g2d.fillOval(70,70,100,100);
}
```

这跟上面差不多，只是渐层颜色是随机挑选的

要点

事件

- GUI从创建window开始，通常会使用JFrame

```
JFrame frame = new JFrame();
```

- 你可以这样加入按钮、文字字段等组件：

```
frame.getContentPane( ).add(button);
```

- JFrame与其他组件不同，不能直接加上组件，要用它的content pane。
- 要显示window，你得指定尺寸和执行显示动作

```
frame.setSize(300, 300);
frame.setVisible(true);
```

- 监听GUI事件才能知道用户对接口做了什么事情。
- 你必须要对事件源注册所要监听的事件。事件源是一种会根据用户操作而触发事件的机制。
- 监听接口让事件源能够调用给你。
- 要对事件源注册就调用事件源的注册方法，它的方法一定是add<EventType>Listener这种形式。以按钮的ActionEvent注册为例：

```
button.addActionListener(this);
```

- 通过实现所有的事件处理方法来实现监听接口。对 ActionEvent 而言，方法可能像这样：

```
public void actionPerformed(ActionEvent
                                event) {
button.setText("Clicked!");
}
```

- 传递给事件处理方法的事件对象带有事件的信息，其中包括了事件源。

图形

- 二维图形可以直接画在图形组件上。
- .gif与.jpeg文件可以直接放在组件上。
- 用JPanel的子类覆盖paintComponent()方法绘制自定义的图形。
- paintComponent()方法会由GUI系统调用，你不可以自己调用。它的参数是个你不能自己创建的Graphics对象。
- Graphics对象有些你可以调用的方法，像是：

```
graphics.setColor(Color.blue);
g.fillRect(20, 50, 100, 120);
```

- 使用Image来绘制.jpg：

```
Image image = new ImageIcon("pic.
jpg").getImage( );
g.drawImage(image, 3, 4, this);
```

- paintComponent()的Graphics参数实际上是个Graphics2D。
- 调用Graphics2D的方法前，你必须把Graphics对象转换为Graphics2D。

```
Graphics2D g2d = (Graphics2D) g;
```

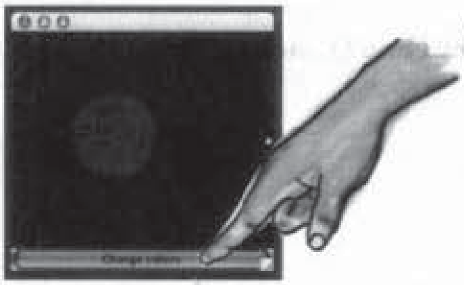
我们可以获得事件，也可以绘制图形。但可以在获得事件时绘制图形吗？

现在让我们试试看在事件发生时改变面板的图案。让圆圈在用户按下按钮时改变颜色。下面列出程序的流程：

启动程序

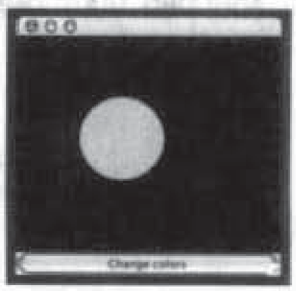


1. 这是个运用到panel和button组件的frame。将监听向按钮注册，然后显示出frame并等待用户点击。



2. 用户点击按钮，因此创建出一个事件对象并调用监听的事件处理程序。

3. 事件处理过程调用frame的repaint()，然后系统会调用panel的paintComponent()。



4. 好了！因为paintComponent()又运行了一次，所以圆圈被填上不同的颜色。

一个frame上面怎么可以摆两个widget吗？



GUI的布局：超过一个以上widget的frame

下一章会讨论GUI的布局（layout），但我们现在可以先快速地看一遍。frame默认有5个区域可以安置widget。每个区域只能安置一项，但是别担心！该项目可以是能够安置包括面板在内的3项东西的面板，所以你可以在面板上面放面板。事实上，我们在安置按钮的时候就作弊了：

```
frame.getContentPane().add(button);
```

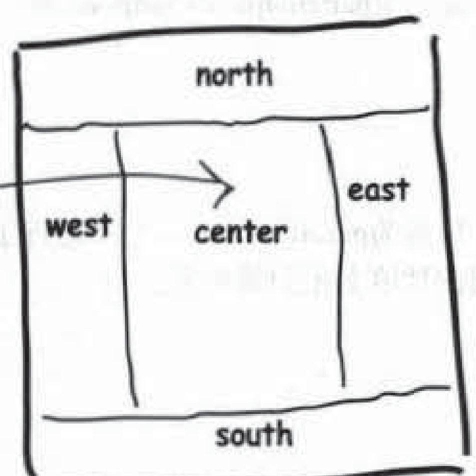
其实你不应该这么做

明确指定加到默认content pane上的位置是比较好的方式

```
frame.getContentPane().add(BorderLayout.CENTER, button);
```

两个参数的add()方法可以指定使用的区域

如果使用的是单一参数的add()方法会自动地把widget加到中心区域

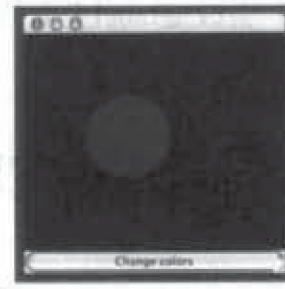


Sharpen your pencil

写出一个程序可以像369页那样把按钮和面板加到frame上。

按下按钮圆圈就会改变颜色

自定义的面板是放在frame的中心区域



按钮安置于SOUTH区域

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleGui3C implements ActionListener {

    JFrame frame;

    public static void main (String[] args) {
       SimpleGui3C gui = new SimpleGui3C();
       gui.go();
    }

    public void go() {
       frame = new JFrame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       JButton button = new JButton("Change colors");
       button.addActionListener(this);
```

把监听加到按钮上

```
       MyDrawPanel drawPanel = new MyDrawPanel();

       frame.getContentPane().add(BorderLayout.SOUTH, button);
       frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
       frame.setSize(300,300);
       frame.setVisible(true);
    }
```

依照指定区域把widget放上去

```
    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}
```

当用户按下按钮时就要求frame重新绘制

```
class MyDrawPanel extends JPanel {

   public void paintComponent(Graphics g) {
     // 填人彩色，见367页
   }

}
```

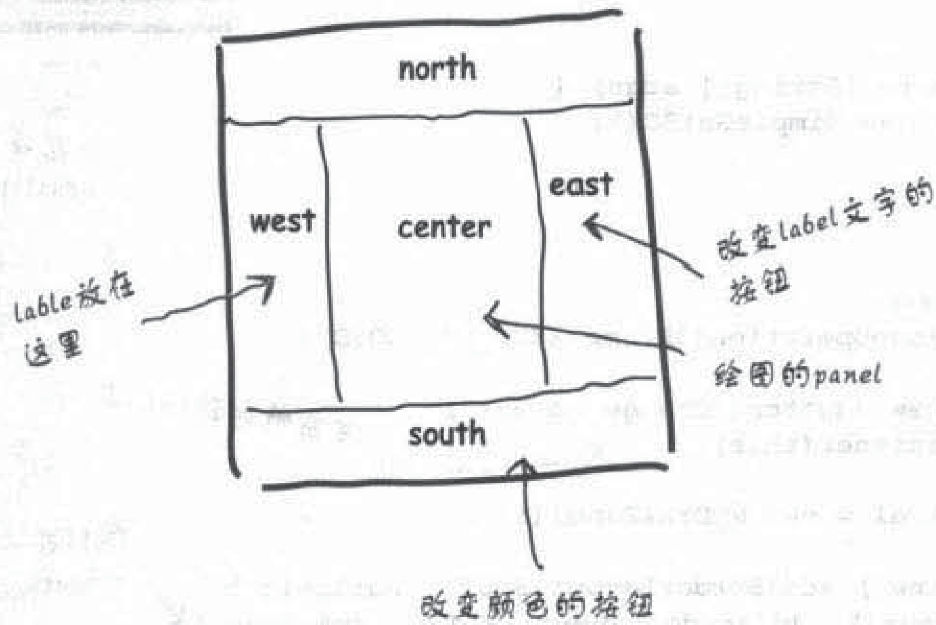
这个方法会在重新绘制frame的时候被调用

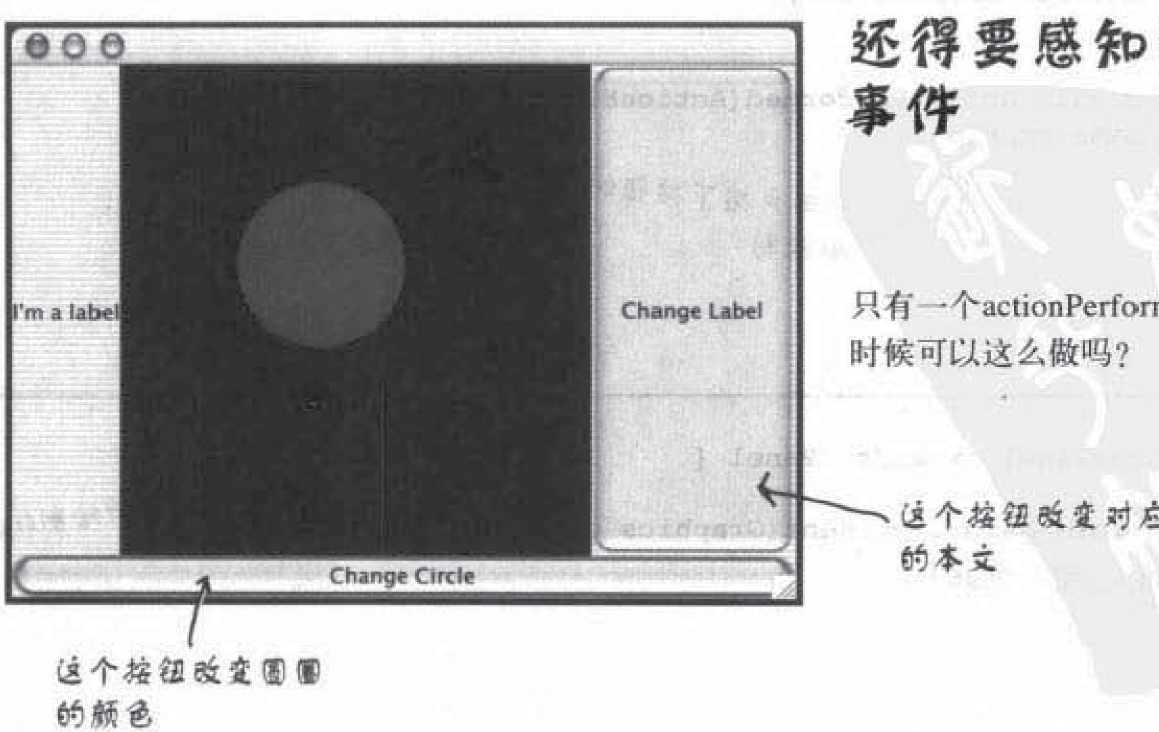
PDG

尝试两个按钮

South的按钮没有改变，还会要求frame重新绘制。第二个按钮（贴在east处）会改变label上面的文字（label是一种显示文字的widget）。

所以现在有4个widget





还得要感知两个事件

只有一个actionPerformed()方法的时候可以这么做吗？

当每个按钮执行不同工作时，要如何对两个不同的按钮分别取得事件？

选项一：实现两个actionPerformed()方法。

```
class MyGui implements ActionListener {
    // 一堆程序代码

    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }

    public void actionPerformed(ActionEvent event) {
        label.setText("That hurt!");
    }
}
```

不可以这样！

不能这么做！ 你不能在实现同一个类的同一个方法两次，这过不了编译这一关。就算可以，事件源怎么分得出要调用哪一个？

选项二：对两个按钮注册同一个监听口。

```
class MyGui implements ActionListener {
    // 声明一组实例变量

    public void go() {
        // 创建GUI
        colorButton = new JButton();
        labelButton = new JButton();
        colorButton.addActionListener(this);
        labelButton.addActionListener(this);
        // 还有一些GUI程序……
    }

    public void actionPerformed(ActionEvent event) {
        if (event.getSource() == colorButton) {
            frame.repaint();
        } else {
            label.setText("That hurt!");
        }
    }
}
```

注册同一个监听口

查询事件对象来看实际上事件是哪个事件源发出的

可以是可以啦，但这看起来不太像面向对象。 用单一的事件处理程序对付不同的东西意味着执行太多不同工作的方法。如果想要改变某个工作，很可能会把全部工作都弄乱。这样解决会对可读性和维护工作产生危害。

当每个按钮执行不同工作时，要如何对两个不同的按钮分别取得事件？

● 选项三：创建不同的ActionListener。

```
class MyGui  {
   JFrame frame;
   JLabel label;
   void gui() {
      // 实际程序代码
   }
} // 关闭类
```

```
class ColorButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}
```

不行！这个类没有对MyGui上frame变量的引用

```
class LabelButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        label.setText("That hurt!");
    }
}
```

有问题！此类没有label的引用变量

这些类没有办法存取到所需的变量。你可以加以改正，但必须要给予每个监听类对GUI类的引用，才能让actionPerformed()方法中的监听能够使用类的引用来存取它的变量。这却又会破坏封装的特性，因此我们或许得需要更好的getter函数（例如getFrame()或getLabel()等），并且你或许也需要对监听类加上一个构造函数以便能够在监听初始化的同时传入GUI的引用。不过这样只会加深混乱和复杂的程度。

一定有比较好的方法吧？


如果可以有两个不同的监听类，它们却能够存取GUI的实例变量，并且两者都好像互相属于对方一样，这该有多好？如果我能够换一个现代一点的发型应该会更好，最好是衣服也换一套，这一套现在看起来有点像是在装可爱……

内部类是我们的救星！

一个类可以嵌套在另一个类的内部。这很简单，只要确定内部类的定义是包在外部类的括号中就可以。

单纯的内部类：

```
class MyOuterClass  {

   class MyInnerClass {
      void go() {
      }
   }

}
```

内部类完全被外部的类包起来

> 内部类可以使用外部所有的方法与变量，就算是私用的也一样。
>
> 内部类把存取外部类的方法和变量当作是开自家冰箱。

内部的类对外部的类有一张特殊的通行证，能够自由地存取它的内容，就算是私用的内容也一样。内部类可以把外部的方法或变量当作是自己的。这就是为何内部的类非常好用的原因，除了跟正常的类没有差别之外，还多了特殊的存取权。

内部类可以使用外部的变量

```
class MyOuterClass  {

    private int x;

   class MyInnerClass {
      void go() {
         x = 42;
      }
   } // 关闭内部类

} // 关闭外部类
```

把x当作自己的！

内部类的实例一定会绑在外部类的实例上*

内部对象与外部对象有发生过超友谊的关系

要记住，当我们讨论内部类可以存取外部类的内容时，意思是说内部类的实例可以存取外部类实例的内容。但是哪个实例呢？

任意一个内部类可以存取其他外部类的方法和变量吗？不行！只能存取它所属的那一个！



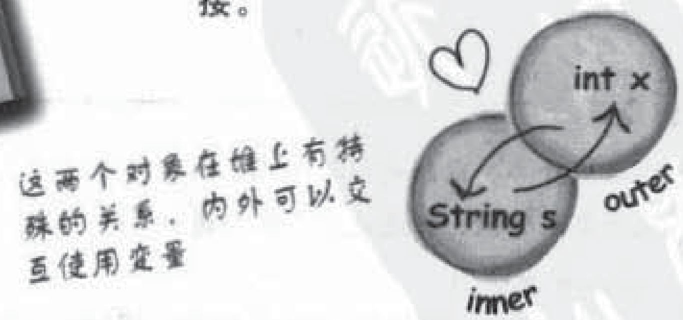
① 创建外部类的实例



② 使用外部类的实例来创建内部类的实例



③ 外部和内部对象有着亲密的连接。



*有一种非常特殊的异常情况——内部类是定义在静态的方法中。本书不打算讨论这个，你也可能一辈子都不会遇到。

如何创建内部类的实例

如果你从外部类程序代码中初始化内部的类，此内部对象会绑在该外部对象上。例如，如果某个方法的程序代码会初始化内部的类，此内部对象会绑在执行该 方法的实例上。

外层类的程序代码可以用初始化其他类完全相同的方法初始它所包容的内部类。

```
class MyOuter  {

   private int x;

   MyInner inner = new MyInner();

  public void doStuff() {
     inner.go();
   }

  class MyInner {
     void go() {
        x = 42;
     }
   } // 关闭内部类

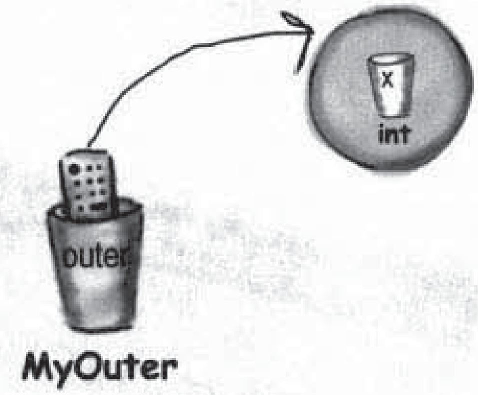
} // 关闭外部类
```

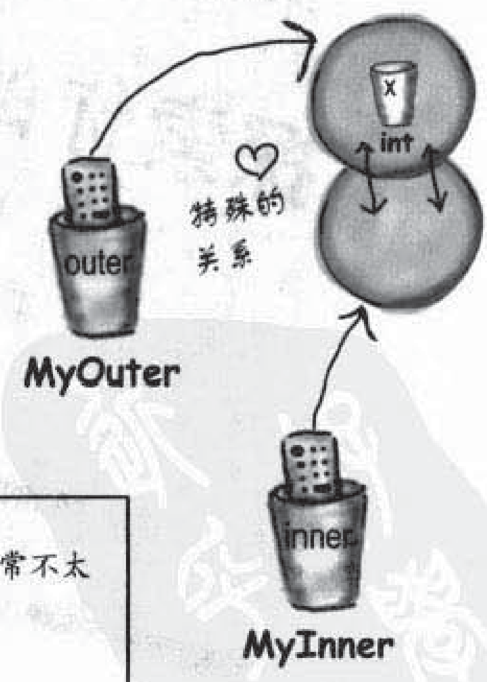
外部有个私用的x实例变量

创建内部的实例

调用内部的方法

内部可以使用外部的x变量



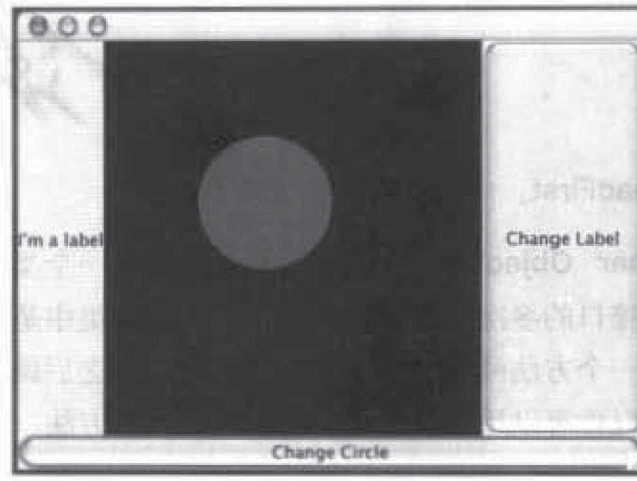


附注

你也可以从外部类以外的程序代码来初始内部实例，但这要使用特殊的语法。通常不太会有机会要这么做，但还是先让你知道一下。

```
class Foo {
   public static void main (String[] args) {
      MyOuter outerObj = new MyOuter();
      MyOuter.MyInner innerObj = outerObj.new MyInner();
   }
 }
```

现在就可以实现两个按钮的程序




```
public class TwoButtons {

    JFrame frame;
    JLabel label;

    public static void main (String[] args) {
       TwoButtons gui = new TwoButtons ();
       gui.go();
    }

    public void go() {
       frame = new JFrame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       JButton labelButton = new JButton("Change Label");
       labelButton.addActionListener(new LabelListener());

       JButton colorButton = new JButton("Change Circle");
       colorButton.addActionListener(new ColorListener());

       label = new JLabel("I'm a label");
       MyDrawPanel drawPanel = new MyDrawPanel();

       frame.getContentPane().add(BorderLayout.SOUTH, colorButton);
       frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
       frame.getContentPane().add(BorderLayout.EAST, labelButton);
       frame.getContentPane().add(BorderLayout.WEST, label);

       frame.setSize(300,300);
       frame.setVisible(true);
    }

     class LabelListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            label.setText("Ouch!");
        }
     } // 关闭内部类

     class ColorListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            frame.repaint();
        }
     }  // 关闭内部类

}
```

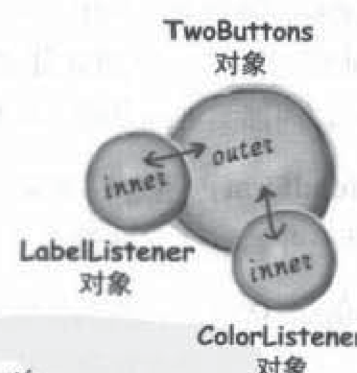
现在主要的GUI类并不实现ActionListener

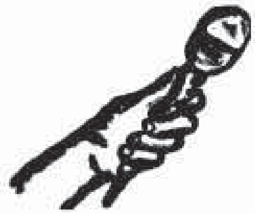
相对于将this传给监听的注册方法，现在传的是对应的实例

终于可以在单一的类中做出不同的ActionListener！

内部可以存取label

直接存取frame，不需要明确指定外部类的引用





Java Exposed

本周的来宾：内部类的实例

HeadFirst：内部类有什么重要？

Inner Object：怎么说呢，我们提供在一个类中实现同一接口的多次机会。要知道，在一般的类中是不能实现同一个方法两次的。但使用了内部的类之后就可以了，所以你可以用不同方法实现同一个接口方法。

HeadFirst：为什么要实现同一个方法两次？

Inner Object：回想一下GUI事件处理程序。如果你想要让3个按钮有不同的事件行为，就要使用3个内层类来个别实现ActionListener，也就是每个类实现自己的actionPerformed()方法。

HeadFirst：所以说事件处理程序是唯一的理由了？

Inner Object：当然不是。这只是个明显的例子。任何时候你需要一个独立却又好像另一个类成员之一的类时，内部类可能是唯一的解。

HeadFirst：还是搞不懂，如果需要独立的类，为什么不一开始就独立地创建？

Inner Object：因为要实现同一个接口好几次。就算不是这样，你也会需要两个不同的类来表示两项不同的事物，这样才是好的面向对象。

HeadFirst：哇！我以为面向对象代表重用与维护呢。两个不同的类可以分开维护，但包在一起不就纠缠住了吗？并且被包起来的类不是就不能重用吗？

Inner Object：没错，被包起来的内部类无法像独立的类一样重用，因为它会与外部的类紧密地结合。但是……

HeadFirst：这就是我说的，如果不能重用又何必这样呢？我是说这根本就是为了解决接口的错误而产生的啊。

Inner Object：就像我已经说过的，你要用IS-A和多态的观点来看这件事。

HeadFirst：可以，这是因为……

Inner Object：因为外部与内部的类需要通过不同的IS-A测试！以GUI的监听为例，按钮的监听声明要注册什么？也就是说要传给addActionListener()什么东西？

HeadFirst：这种情况下要传进一个ActionListener。你的重点是什么？

Inner Object：重点在于polymorphically，有个方法只能采用特定的类型。有时这可以通过ActionListener的IS-A测试，但最重要的是，如果你的类必须IS-A别的类呢？

HeadFirst：为什么不让你的类去继承该类呢？

Inner Object：如果它本来就已经继承不相干的类呢？

HeadFirst：噢，我明白了，接口的实现可以超过一个，但类仅能继承一个而已。

Inner Object：很好！没错，你不能同时又是Dog又是按钮，但有时又必须这样。Dog可以继承Animal却有个内部的类来代表按钮的行为，因此在有需要的时候Dog就可以派出内部的类来代表按钮。也就是说Dog虽然不能x.takeButton(this)但是可以x.takeButton(new DogInnerButton())。

HeadFirst：可不可以再举个更清楚的例子？

Inner Object：还记得JPanel的子类吗？现在我们还把它写成独立的类。这还好，因为此类不需要对GUI的实例变量有特殊的存取权。但若是要呢？如果我们要让这个面板存取主程序的坐标变量来执行动画呢？在这种情况下，我们可以让绘图的面板成为内部的类，且是JPanel的子类，而外部类就可以自由地去当别的类的子类。

HeadFirst：我懂了！还有，绘图的面板也不是那么可以独立的重用，因为它实际上只是写给特定的GUI程序用。

Inner Object：很好，这块饼干给你吃。

HeadFirst：接下来我们可以继续踢爆你与外部实体间的亲密关系。

Inner Object：你们这些人是怎么了？有线电视的新闻台看太多了是不是？

HeadFirst：唉呦，你又不是不知道观众最喜欢听八卦。所以有人把你创建出来之后你就算是它“包养”的是吧？

Inner Object：这倒是真的，有人把这当作是一种婚约。

HeadFirst：就说是婚约吧。你们能够离婚再嫁吗？

Inner Object：不行，这是一辈子的事情。

HeadFirst：谁的一辈子？外部吗？

Inner Object：我自己的。我不能绑在其他外部对象上。唯一的解脱只有garbage collection。

HeadFirst：那外部对象呢？它可以绑其他的内部对象吗？

Inner Object：它可以。而且是同时间进行的，香港人叫它“一王多后”，满意了吧？

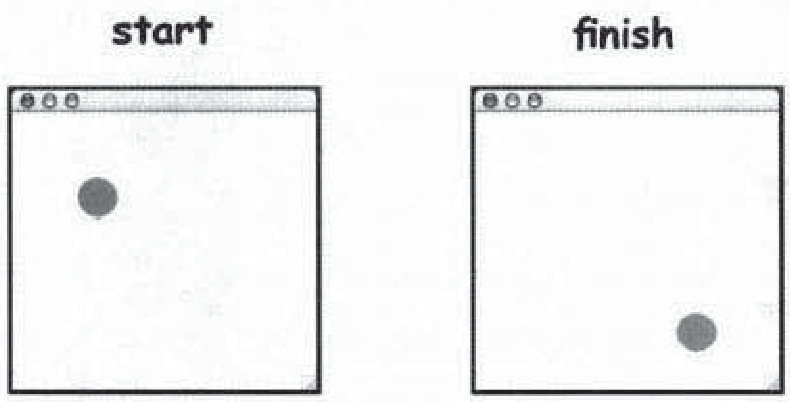
HeadFirst：是你自己先开始多重实现的话题啊。所以外部类同时有多个按钮要多个内部类来照顾事件是很合理的。谢谢，今天的访谈到此结束。



以内部类执行动画效果

我们已经看过为何内部类对事件的监听是很方便的，因为你会对相同的事件处理程序实现一次以上。现在让我们来看看当内部类被用来当作某种外部类无法继承的子类时是多么好用。换句话说，内外部的类可以搞定不同的继承层次！

我们的目标是要创建出简单的动画，让圆圈从画面左上方移动到右下方。



动画效果是如何运动的：

① 在特定坐标点绘制对象。

```
g.fillOval(20,50,100,100);
```

离左上方20，50 个像素

② 在不同的坐标点重新绘制对象。

```
g.fillOval(25,55,100,100);
```

离左上方25，55个像素（稍微向右下方移动）

③ 在坐标尚未到达终点前重复上列步骤。

there are no Dumb Questions

问： 为什么要学动画？我又不想开发游戏。

答： 你也许不会去开发游戏，但可能会碰到仿真器，它也会持续地显示处理运算的结果或者你会需要创建出持续更新的图形来显示内存的耗用状况。这一类的事情都会遇到类似的处理方法。

其实说穿了这只是个展示内部类功用最简单的方式。

其实我们真正需要的是这样……

```
class MyDrawPanel extends JPanel {
      public void paintComponent(Graphics g) {
         g.setColor(Color.orange);
         g.fillOval(x,y,100,100);
      }
}
```

每次paintComponent()被调用时，把图形画在不同的位置上

Sharpen your pencil

但是要如何取得新的坐标呢？
又是谁要来调用repaint()？

你是否能够自己设计出一个简单的解决方案让图形从画面的左上方移动到右下方？答案在下一页，自己还没有想过之前不要偷看！

提示一：把绘图的面板当作是内部的类。

提示二：别在paintComponent()里面放任何种类的循环。

把答案写下来：

完整的动画程序

```
import javax.swing.*;
import java.awt.*;


public class SimpleAnimation {

    int x = 70;
    int y = 70;
```

在主要的GUI中创建两个实例变量用来代表图形的坐标

```
    public static void main (String[] args) {
        SimpleAnimation gui = new SimpleAnimation ();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        MyDrawPanel drawPanel = new MyDrawPanel();
```

此处跟前面一样创建出frame上的widget

```
        frame.getContentPane().add(drawPanel);
        frame.setSize(300,300);
        frame.setVisible(true);
```

这里是重点！

重复130次

```
        for (int i = 0; i < 130; i++) {

            x++;
            y++;
```

递增坐标值

```
            drawPanel.repaint();
```

要求重新绘制面板

```
            try {
              Thread.sleep(50);
            } catch(Exception ex) { }
        }
```

加上延迟刻意放慢，不然一下就会跑完，第15章会说明线程

```
    }// 关闭go()方法
```

内部类

```
    class MyDrawPanel extends JPanel {

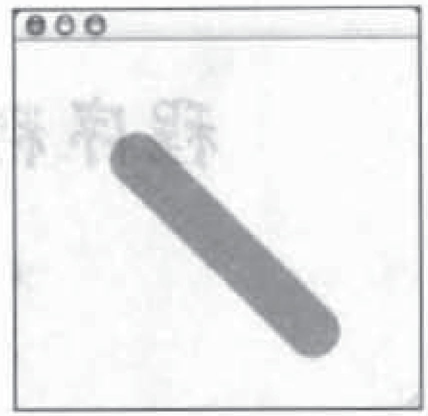
        public void paintComponent(Graphics g) {
            g.setColor(Color.green);
            g.fillOval(x,y,40,40);
```

使用外部的坐标值来更新

```
        }
    } // 关闭内部类
} // 关闭外部类
```

哎呀！它留下了痕迹

哪里弄错了？这个程序有个 bug。



这跟我们想的有点出入

我们忘记擦掉原来的图形，所以会有痕迹

解决的办法是在每次画上新的圆圈之前把整个面板用原来的背景底色填满。下面的程序代码在方法的前面加上两行指令：先把颜色设定为白色，然后填满整个方块区域。也就是说从0,0位置开始以白色填入面板长宽大小的区域。

```
public void paintComponent(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0,0,this.getWidth(), this.getHeight());

    g.setColor(Color.green);
    g.fillOval(x,y,40,40);

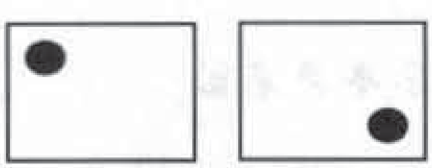
}
```

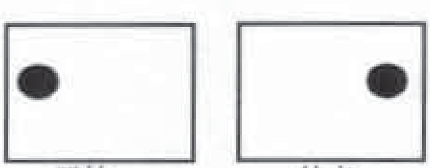
从JPanel继承下来的方法

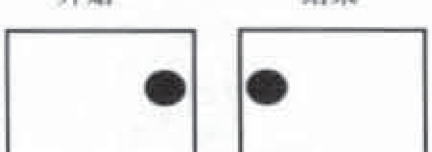


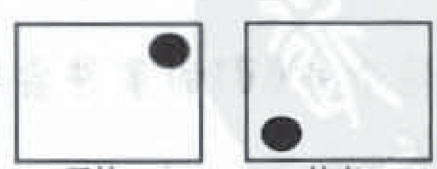
Sharpen your pencil（有空的时候挑战看看）

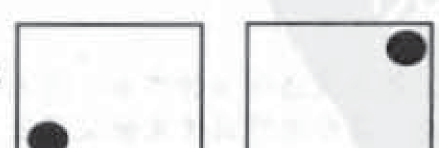
你可以怎样修改坐标的递增值来产生下面所示不同的效果？

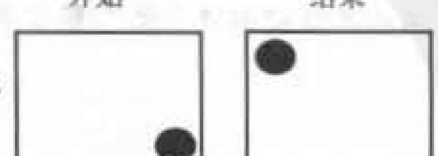
1 
开始 结束
X +3
Y +3

2 
开始 结束
X ____
Y ____

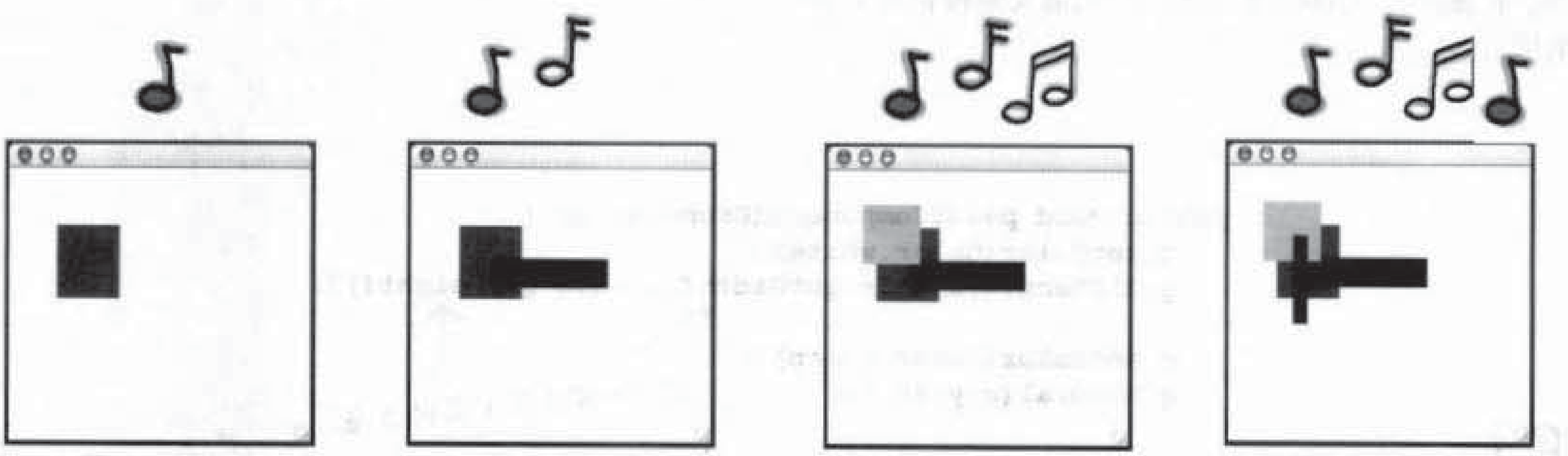
3 
开始 结束
X ____
Y ____

1 
开始 结束
X ____
Y ____

2 
开始 结束
X ____
Y ____

3 
开始 结束
X ____
Y ____

程序料理



让我们来制作音乐录像带！使用Java随机产生的图形来跟着节奏起舞。

这个非GUI的事件会向事件源注册而由音乐本身来触发。

这部分是选择性的内容。但我们认为很值得一看。你应该会很喜欢，并且也可以拿来向父母女友炫耀一番……

（反正不管你做什么他们都会夸奖的）

监听非GUI的事件

对啦，这比不上音乐录像带，但我们还是会做出一个随着音乐节奏绘制随机图形的程序。简单地说，这个程序会监听音乐节奏并在每个拍子上画出随机的方块图形。

这会带来新的问题。目前为止我们只监听过GUI的事件，但现在则需要监听特定类型的MIDI事件。监听非GUI事件的最后结果就跟监听GUI事件是一样的：你会实现出监听者的接口，向事件源注册，然后等待事件源调用你的事件处理程序（定义在监听者的接口中的方法）。

监听音乐节奏的最简单方式是注册并监听实际的MIDI事件，因此只要sequencer收到事件，我们的程序也会取得并绘制图形。但是……有个问题。实际上有个bug会让我们无法监听我们自己制造的MIDI事件（NOTE ON）。

所以我们得做一点小小的修正。我们可以监听另外一种类型的MIDI事件，它被称为ControllerEvent。我们的解决方案是注册ControllerEvent，然后确保每个NOTE ON事件都有对应的ControllerEvent事件会在同一拍上面触发。要怎样确保这件事呢？如同其他事件一样把它加到 track 上！也就是说，我们的sequence会像下面这样：

BEAT 1 - NOTE ON, CONTROLLER EVENT
BEAT 2 - NOTE OFF
BEAT 3 - NOTE ON, CONTROLLER EVENT
BEAT 4 - NOTE OFF……

如此继续下去。

这个音乐艺术程序需要下列功能：

- 制作一系列的MIDI信息/事件来播放任意的钢琴音（或是你自行设定的其他乐器）。
- 对事件注册一个监听者。
- 开始sequecer的播放操作。
- 每当监听者的事件处理程序被调用时，在面板上面画一个随机的方块并调用reapint。

制作程序的3个方式：

- 第一版：简单地制作出MIDI事件，因为要做出很多个。
- 第二版：注册并监听事件，但没有图形。从命令栏对每一拍输出一个信息。
- 第三版：最终版本。在第二版上加上图形输出。

制作信息/事件的简单（偷懒）方法

制作出信息和事件并把它们加到track上是一件很枯燥的工作。对每个信息都得做出信息的实例（ShortMessage）、调用 setMessage()、制作MidiEvent，然后将事件加到track上。上一章的程序代码中我们对每个信息逐步执行这些操作。如此需要8行程序才能做出一个信息！4行做NOTE ON事件、再4行做NOTE OFF事件。

```
ShortMessage a = new ShortMessage();
a.setMessage(144, 1, note, 100);
MidiEvent noteOn = new MidiEvent(a, 1);
track.add(noteOn);

ShortMessage b = new ShortMessage();
b.setMessage(128, 1, note, 100);
MidiEvent noteOff = new MidiEvent(b, 16);
track.add(noteOff);
```

每个事件都要有的操作：

1. 创建信息实例。

```
ShortMessage first = new ShortMessage();
```

2. 调用setMessage()。

```
first.setMessage(192, 1, instrument, 0)
```

3. 制作信息的MidiEvent实例。

```
MidiEvent noteOn = new MidiEvent(first, 1);
```

4. 把事件加到track上。

```
track.add(noteOn);
```

创建静态的实用方法来制作信息并返回MidiEvent

这4个参数是给信息用的

tick是何时播放信息

```
public static MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
        ShortMessage a = new ShortMessage();
        a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);

    }catch(Exception e) { }
    return event;
}
```

哇！5个参数！

使用参数来创建信息

返回事件

范例：如何使用静态的makeEvent()方法

此处没有事件处理或绘图，只有15个攀升的音阶组成的队列。这个程序代码的重点是学习如何使用makeEvent()方法。下两版的程序代码会比较小也比较简单。

```
import javax.sound.midi.*;  // 别忘记要import

public class MiniMusicPlayer1 {

   public static void main(String[] args) {

      try {

         Sequencer sequencer = MidiSystem.getSequencer();  // 创建并打开队列
         sequencer.open();


         Sequence seq = new Sequence(Sequence.PPQ, 4);  // 创建队列并track
         Track track = seq.createTrack();

      for (int i = 5; i < 61; i+= 4) {  // 创建一堆连续的音符事件


         track.add(makeEvent(144,1,i,100,i));  // 调用makeEvent()来产生信息和事件
         track.add(makeEvent(128,1,i,100,i + 2));  // 然后把它们加到track上

       } // 结束循环

          sequencer.setSequence(seq);
          sequencer.setTempoInBPM(220);  // 开始播放
          sequencer.start();
      } catch (Exception ex) {ex.printStackTrace();}
  } // 关闭main


   public static MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
         MidiEvent event = null;
         try {
           ShortMessage a = new ShortMessage();
           a.setMessage(comd, chan, one, two);
          event = new MidiEvent(a, tick);

         }catch(Exception e) { }
        return event;
      }
} // 关闭类
```

第二版：注册并取得ControllerEvent方法

```
import javax.sound.midi.*;
public class MiniMusicPlayer2 implements ControllerEventListener {

    public static void main(String[] args) {
        MiniMusicPlayer2 mini = new MiniMusicPlayer2();
        mini.go();
    }
  public void go() {

     try {
        Sequencer sequencer = MidiSystem.getSequencer();
        sequencer.open();

        int[] eventsIWant = {127};
        sequencer.addControllerEventListener(this, eventsIWant);

        Sequence seq = new Sequence(Sequence.PPQ, 4);
        Track track = seq.createTrack();

        for (int i = 5; i < 60; i+= 4) {
           track.add(makeEvent(144,1,i,100,i));

           track.add(makeEvent(176,1,127,0,i));

           track.add(makeEvent(128,1,i,100,i + 2));
        } // 结束循环

        sequencer.setSequence(seq);
        sequencer.setTempoInBPM(220);
        sequencer.start();
     } catch (Exception ex) {ex.printStackTrace();}
  } // 关闭


   public void controlChange(ShortMessage event) {
        System.out.println("la");
   }

  public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
        MidiEvent event = null;
        try {
          ShortMessage a = new ShortMessage();
          a.setMessage(comd, chan, one, two);
          event = new MidiEvent(a, tick);

        }catch(Exception e) { }
        return event;
     }
} // 关闭类
```

我们必须要监听ControllerEvent，因此实现了这个接口

向sequencer注册事件。注册的方法取用监听者与代表想要监听的事件的int数组，我们只需要127事件

插入事件编号为127的自定义ControllerEvent（176），它不会做任何事情，只是让我们知道有音符被播放，因为它的tick跟NOTE ON是同时进行的

获知事件时在命令行打印出字符串的事件处理程序

程序代码与上一版不同之处用底色强调出来

第三版：与音乐同步输出图形

最终版本是用第二版加上GUI部分而成的。我们创建出frame，加上绘图的面板，并在取得事件的同时画出新的方块并要求重绘画面。另外一个改变的地方是从连续攀升变成随机产生的音符。

除了简单的GUI之外，最重要的程序变化在于让绘图面板实ControllerEventListener而不是由程序本身来实现。因此当内部类所做的绘图板获知事件时，它会知道该做些什么事。

完整的程序代码列在下一页。

绘图面板的内部类

这个面板也是监听者

```
class MyDrawPanel extends JPanel implements ControllerEventListener {

     boolean msg = false; <— 获知事件时才会为真

     public void controlChange(ShortMessage event) {
        msg = true; <—
        repaint(); <— 获知事件时设为真并调用重绘程序
     }

     public void paintComponent(Graphics g) {
      if (msg) { <— 因为也有其他东西会引发重绘，所以要判断是否为
                     ControllerEvent所引发的

        Graphics2D g2 = (Graphics2D) g;

        int r = (int) (Math.random() * 250);
        int gr = (int) (Math.random() * 250);
        int b = (int) (Math.random() * 250);

        g.setColor(new Color(r,gr,b));

        int ht = (int) ((Math.random() * 120) + 10);
        int width = (int) ((Math.random() * 120) + 10);
        int x = (int) ((Math.random() * 40) + 10);
        int y = (int) ((Math.random() * 40) + 10);
        g.fillRect(x, y, widtho, ht);
        msg = false;

      } // if结束
    } // 关闭方法
  } // 关闭内部类
```

其余的程序代码是在产生随机的颜色并画出方块

Sharpen your pencil

这是第三版程序代码的完整列表。它是直接使用第二版来改造的。试着不偷看前面的内容来自己加上注释，偷看最可耻。

```
import javax.sound.midi.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;

public class MiniMusicPlayer3 {

    static JFrame f = new JFrame("My First Music Video");
    static MyDrawPanel ml;

    public static void main(String[] args) {
          MiniMusicPlayer3 mini = new MiniMusicPlayer3();
          mini.go();
    } // 关闭方法


    public  void setUpGui() {
       ml = new MyDrawPanel();
       f.setContentPane(ml);
       f.setBounds(30,30, 300,300);
       f.setVisible(true);
    } // 关闭方法

    public void go() {
       setUpGui();

       try {

          Sequencer sequencer = MidiSystem.getSequencer();
          sequencer.open();
          sequencer.addControllerEventListener(ml, new int[] {127});
          Sequence seq = new Sequence(Sequence.PPQ, 4);
          Track track = seq.createTrack();

          int r = 0;
          for (int i = 0; i < 60; i+= 4) {

             r = (int) ((Math.random() * 50) + 1);
             track.add(makeEvent(144,1,r,100,i));
             track.add(makeEvent(176,1,127,0,i));
             track.add(makeEvent(128,1,r,100,i + 2));
          } // 结束循环

          sequencer.setSequence(seq);
          sequencer.setTempoInBPM(120);
          sequencer.start();
       } catch (Exception ex) {ex.printStackTrace();}
    } // 关闭方法
```

```
   public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
       MidiEvent event = null;
       try {
          ShortMessage a = new ShortMessage();
          a.setMessage(comd, chan, one, two);
           event = new MidiEvent(a, tick);

        }catch(Exception e) { }
        return event;
    } // 关闭方法


 class MyDrawPanel extends JPanel implements ControllerEventListener {
      boolean msg = false;

      public void controlChange(ShortMessage event) {
         msg = true;
         repaint();
      }

      public void paintComponent(Graphics g) {
       if (msg) {

         Graphics2D g2 = (Graphics2D) g;

         int r = (int) (Math.random() * 250);
         int gr = (int) (Math.random() * 250);
         int b = (int) (Math.random() * 250);

         g.setColor(new Color(r,gr,b));

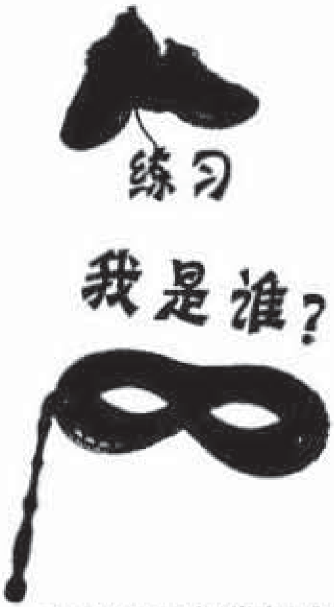
         int ht = (int) ((Math.random() * 120) + 10);
         int width = (int) ((Math.random() * 120) + 10);

         int x = (int) ((Math.random() * 40) + 10);
         int y = (int) ((Math.random() * 40) + 10);

         g.fillRect(x,y,ht, width);
         msg = false;

       } // close if
     } // 关闭方法
   }  // 关闭内部类

} // 关闭类
```



一组Java组件精心打扮出席化装舞会，中场时间有人提议要玩猜猜我是谁的游戏，你可以根据它们对自己的描述来猜测出提示的是哪位。规则是每个组件都得说实话，若某些提示同时对数个组件都为真的话，则将它们全部填入。

今晚出席舞会的有：

这一章提到过的每个家伙都有可能出现！

整个GUI都掌握在我的手上。 ____________________

每个事件类型都有一个。 ____________________

监听者的关键方法。 ____________________

这个方法会设定JFrame的大小。 ____________________

你会帮这个方法写程序代码，但是不会调用它。 ____________________

当用户确实做了某个操作之后，它就会发生。 ____________________

大部分都是事件源。 ____________________

把数据带回监听者。 ____________________

addXxxListener()会说对象是个…… ____________________

监听者是如何注册的？ ____________________

所有绘图程序代码的去处。 ____________________

通常会绑定在某个实例上。 ____________________

(Graphic g)的这个g，其实是个…… ____________________

推动paintComponent()的那只手。 ____________________

大部分Swing呆的地方。 ____________________



练习

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class InnerButton {

  JFrame frame;
  JButton b;

  public static void main(String [] args) {
    InnerButton gui = new InnerButton();
    gui.go();
  }

  public void go() {
    frame = new JFrame();
    frame.setDefaultCloseOperation(
                  JFrame.EXIT_ON_CLOSE);

    b = new JButton("A");
    b.addActionListener();

    frame.getContentPane().add(
                  BorderLayout.SOUTH, b);
    frame.setSize(200,100);
    frame.setVisible(true);
  }

  class BListener extends ActionListener {
    public void actionPerformed(ActionEvent e) {
      if (b.getText().equals("A")) {
        b.setText("B");
      } else {
        b.setText("A");
      }
    }
  }
}
```

我是编译器



这左边的Java程序代码代表一份完整的源文件。你的任务是要扮演编译器角色并判断这支程序是否可以编译过关。如果有问题，哪里要修改？如果没问题，那它是做什么的？

泳池迷宫

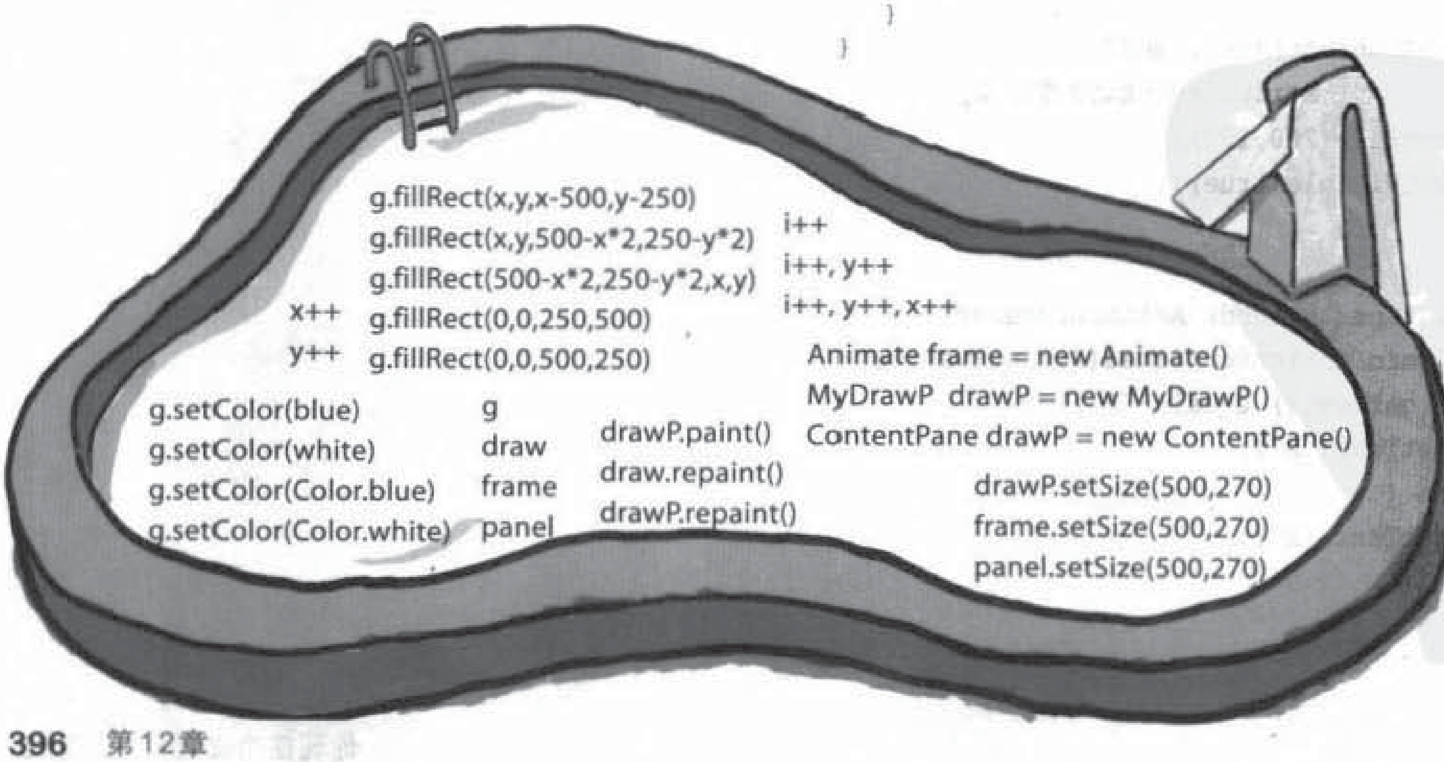


你的任务是要从游泳池中挑出程序片段并将它填入右边的空格中。同一个片段有可能使用两次以上，且泳池中有些多余的片段。填完空格的程序必须能够编译与执行并产生出下面的输出。

输出：一个完美、了不起的蓝色方块，还会慢慢变小变白。



```
import javax.swing.*;
import java.awt.*;
public class Animate {
    int x = 1;
    int y = 1;
    public static void main (String[] args) {
       Animate gui = new Animate ();
       gui.go();
    }
    public void go() {
       JFrame _________ = new JFrame();
       frame.setDefaultCloseOperation(
                     JFrame.EXIT_ON_CLOSE);
       ______________________________________;
       _________.getContentPane().add(drawP);
       ________________________________;
       _________.setVisible(true);
       for (int i=0; i<124; _____________) {
       _______________________;
       _______________________;
       try {
          Thread.sleep(50);
       } catch(Exception ex) { }
       }
    }
    class MyDrawP extends JPanel {
       public void paintComponent (Graphic
                                      _______) {
          _______________________________;
          _______________________________;
          _______________________________;
          _______________________________;
       }
    }
}
```



练习解答

我是谁?

JFrame

Listener interface

actionPerformed( )

setSize( )

paintComponent( )

event

swing component

event object

event source

addActionListener( )

paintComponent( )

inner class

Graphics2D

repaint( )

javax.swing

我是编译器

修好之后，这个程序会产生一个按钮并在点选的时候在A与B之间切换

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

class InnerButton {

  JFrame frame;
  JButton b;

  public static void main(String [] args) {
    InnerButton gui = new InnerButton();
    gui.go();
  }

  public void go() {
    frame = new JFrame();
    frame.setDefaultCloseOperation(
                       JFrame.EXIT_ON_CLOSE);

    b = new JButton("A");
    b.addActionListener( new BListener() );

    frame.getContentPane().add(
                       BorderLayout.SOUTH, b);
    frame.setSize(200,100);
    frame.setVisible(true);
  }

  class BListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      if (b.getText().equals("A")) {
        b.setText("B");
      } else {
        b.setText("A");
      }
    }
  }
}
```

泳池迷宫



```
import javax.swing.*;
import java.awt.*;
public class Animate {
    int x = 1;
    int y = 1;
    public static void main (String[] args) {
       Animate gui = new Animate ();
       gui.go();
    }
    public void go() {
       JFrame frame = new JFrame();
       frame.setDefaultCloseOperation(
                  JFrame.EXIT_ON_CLOSE);
       MyDrawP drawP = new MyDrawP();
       frame.getContentPane().add(drawP);
       frame.setSize(500,270);
       frame.setVisible(true);
       for (int i = 0; i < 124; i++,x++,y++ ) {
          x++;
          drawP.repaint();
          try {
            Thread.sleep(50);
          } catch(Exception ex) { }
       }
    }
    class MyDrawP extends JPanel {
       public void paintComponent(Graphics g ) {
            g.setColor(Color.white);
            g.fillRect(0,0,500,250);
            g.setColor(Color.blue);
            g.fillRect(x,y,500-x*2,250-y*2);
       }
    }
}
```

13 swing

运用Swing



Swing真的很简单。虽然看起来没什么难度，等到执行时你会发现“位置怎么跑掉了？”很容易写也会很难控制的原因在于“布局管理器”。这个对象可以控制 Java的GUI上的widget的大小与位置。它帮你做了很多事情，但并不一定是你想要的结果。你想让两个按钮保持同样的大小，但却没有。打算让文字字段的长度保持在3寸长，结果却是9寸。但是只要稍微运作一下，就可以让布局管理器按照你的想法执行。这一章会讨论Swing和布局管理器，以及更多有关widget的事情。

Swing的组件

组件（component，或称元件）是比我们之前所称的widget更为正确的术语。它们就是你会放在 GUI 上面的东西。这些东西是用户会看到并与其交互的，像是Text Field、button、scrollable list、radio button等。事实上所有的组件都是继承自javax.swing.JComponent。

> 从技术上来说，widget是个Swing的组件，几乎所有的GUI组件都来自于java.swing.JComponent。

组件是可以嵌套的

在Swing中，几乎所有组件都能够安置其他的组件。也就是说，你可以把任何东西放在其他东西上。但在大部分的情况下，你会把像是按钮或列表等用户交互组件放在框架和面板等背景组件上。

除了JFrame之外，交互组件与背景组件的差异不太明确。举例来说JPanel通常用在背景上，但是也可以与用户交互。就跟其他组件一样，你也可以向JPanel注册鼠标的点选等事件。

创建GUI四个步骤的回顾

① 创建window（JFrame）。

```
JFrame frame = new JFrame();
```

② 创建组件。

```
JButton button = new JButton("click me");
```

③ 把组件加到frame上。

```
frame.getContentPane().add(BorderLayout.EAST, button);
```

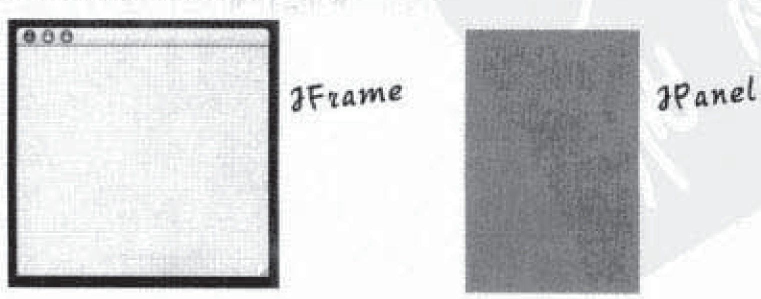
④ 显示出来。

```
frame.setSize(300,300);
frame.setVisible(true);
```

把组件：

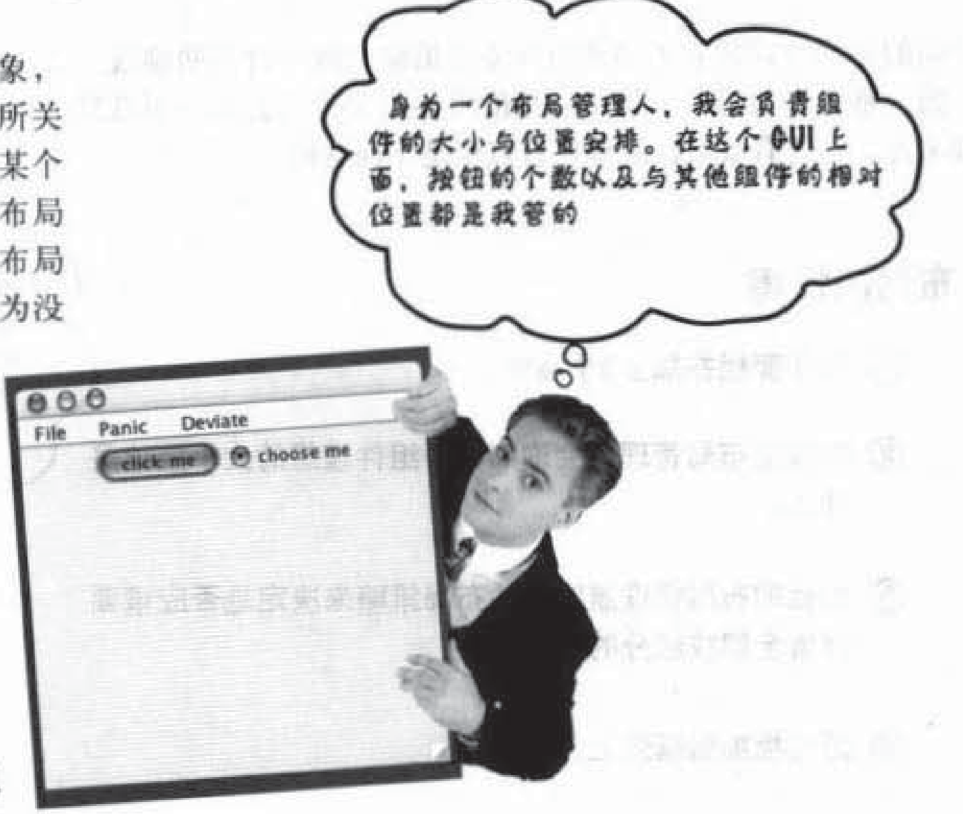


放到背景组件上：



布局管理器（Layout Managers）

布局管理器是个与特定组件相关联的Java对象，它大多数是背景组件。布局管理器用来控制所关联组件上携带的其他组件。也就是说，如果某个框架带有面板，而面板带有按钮，则面板的布局管理器控制着按钮的大小与位置，而框架的布局管理器则控制着面板的大小与位置。按钮因为没有携带其他组件，所以不需要布局管理器。



如果面板带有5项组件，就算这5项都有自己的布局管理器，它们的大小与位置都还是由面板的布局管理器来管理。

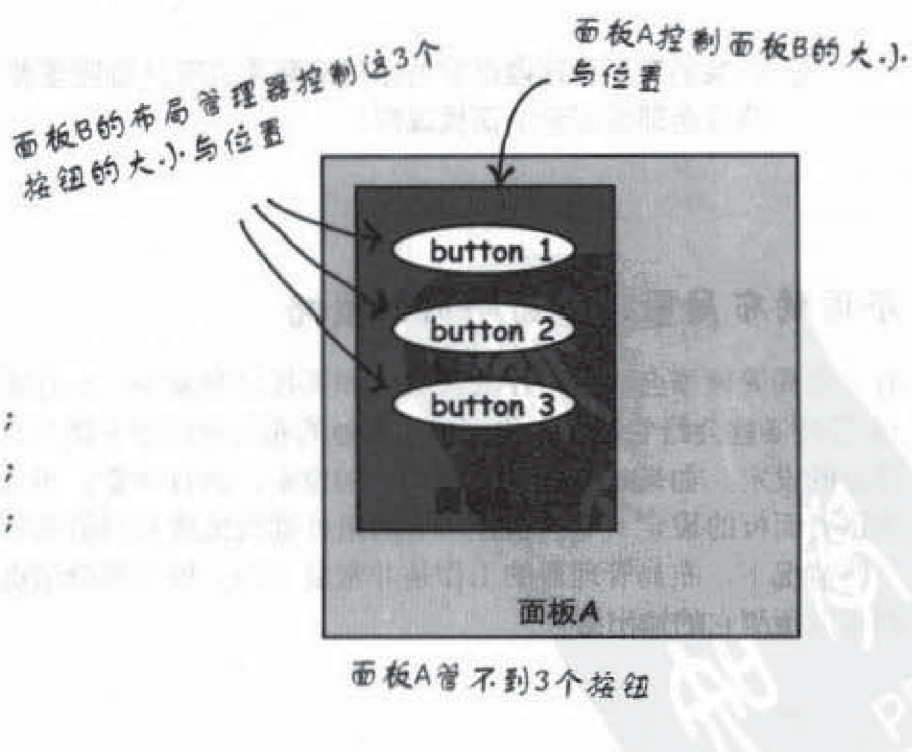
携带的意思就是加入到上面的，面板携带按钮就是因为按钮像下面这样被加到面板上。

```
myPanel.add(button);
```

布局管理器有几种不同的类型，每个背景组件都可以有自定义规则的布局管理器。例如某个布局管理器会让所有的面板维持相同的大小，而另一个布局管理器会让组件自行设定大小，但却又垂直对齐。

下面是嵌套布局的例子：

```
JPanel panelA = new JPanel();
JPanel panelB = new JPanel();
panelB.add(new JButton("button 1"));
panelB.add(new JButton("button 2"));
panelB.add(new JButton("button 3"));
panelA.add(panelB);
```



布局管理器是如何做决定的？

不同的布局管理器有不同的组件安置策略（例如对齐网格线、大小一致、垂直堆放等）。但被安排的组件至少可以表达一下意见。一般来说，处理背景组件的程序有点像是下面这样。

布局的情境

① 制作面板并加上3个按钮。

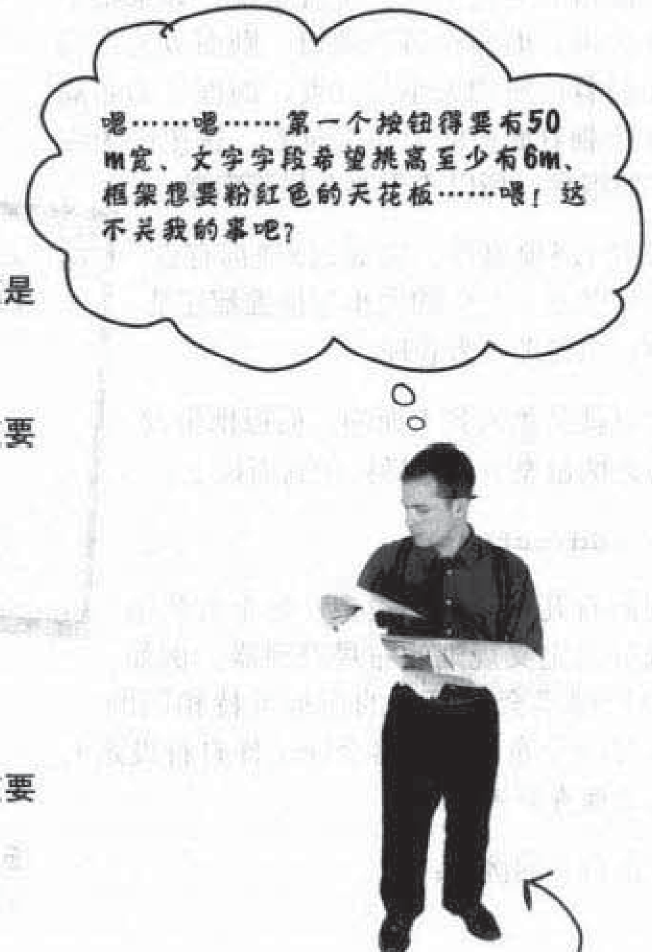
② 面板的布局管理器会询问每个组件理想的大小应该是什么。

③ 面板的布局管理器以它的布局策略来决定是否应该要尊重全部或部分的按钮理想。

④ 把面板加到框架上。

⑤ 框架的布局管理器询问面板的理想尺寸。

⑥ 框架的布局管理器以它的布局策略来决定是否应该要尊重全部或部分的面板理想。



传说中的布局管理员

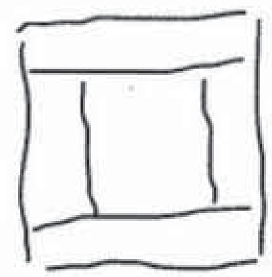
不同的布局管理器有不同的策略

有些布局管理器会尊重组件的想法。如果按钮想要30 × 50像素，布局 管理器就会给它这么大的面积。其他的布局管理器可能只会尊重部分的设定。如果此时按钮想要30 × 50像素，会有30宽，但高度得要跟着面板的设定。有些会让所有的组件都设定成相同的宽度。在某些情况下，布局管理器的工作是非常复杂的。但大部分情况下你都可以预测它的输出结果。

世界三大首席管理器：border、flow和box

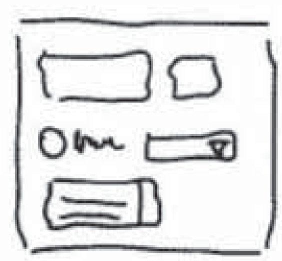
BorderLayout

这个管理器会把背景组件分割成5个区域。每个被管理的区域只能放上一个组件。由此管理员安置的组件通常不会取得默认的大小。这是框架默认的布局管理器！



FlowLayout

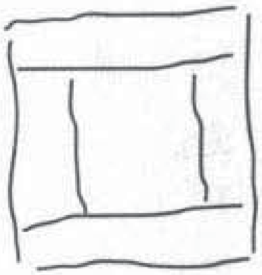
这个管理器的行为跟文书处理程序的版面配置方式差不多。每个组件都会依照理想的大小呈现，并且会从左到右依照加入的顺序以可能会换行的方式排列。因此在组件放不下的时候会被放到下一行。这是面板默认的布局管理器！



BoxLayout

它就像FlowLayout一样让每个组件使用默认的大小，并且按照加入的顺序来排列。但BoxLayout是以垂直的方向来排列（也可以水平，但通常我们只在乎垂直方式）。不像FlowLayout会自动地换行，它让你插入某种类似换行的机制来强制组件从新的一行开始排列。





Borderlayout布局的5个区域：
东区、西区、北区、南区与中央区

将一个按钮加入东区：

```
import javax.swing.*;
import java.awt.*;    ← 它在java.awt这个包里面

public class Button1 {

    public static void main (String[] args) {
        Button1 gui = new Button1();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        JButton button = new JButton("click me");      ↙ 指定区域
        frame.getContentPane().add(BorderLayout.EAST, button);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```

Brain Barbell

(1) BorderLayout是如何设定按钮的大小？

(2) 有哪些因素是必须考虑的？

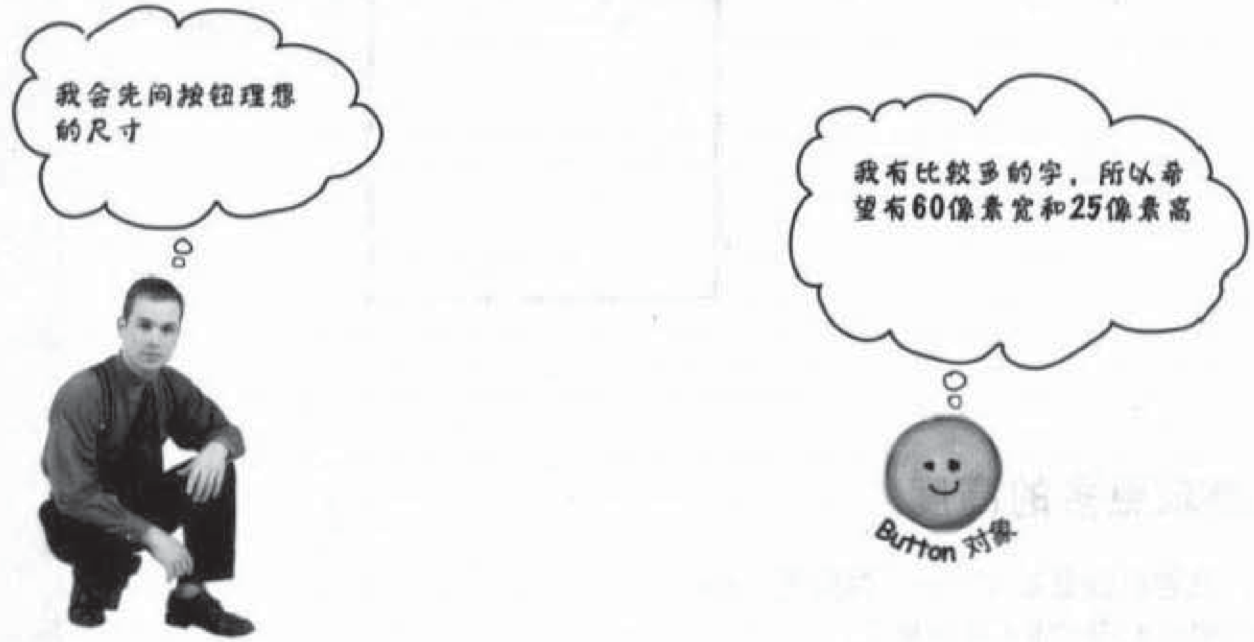
(3) 它为什么不会更宽或更高？



注意按钮字变多时所发生的事……

只有改变按钮的文字

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("click like you mean it");
    frame.getContentPane().add(BorderLayout.EAST, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```





尝试进驻北方

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("There is no spoon...");
    frame.getContentPane().add(BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```



按钮的高度跟默认一样，但与frame同宽

让按钮要求更多的高度

怎么做？按钮已经是最宽了——跟框架一样，但我们可以用更大的字体来让它更高。

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("Click This!");
    Font bigFont = new Font("serif", Font.BOLD, 28);
    button.setFont(bigFont);
    frame.getContentPane().add(BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```

更大的字体会强迫框架留更多的高度给按钮



宽度还是一样，但更高了



那中间区域会发生什么事？

中间区域只能捡剩下的（稍后介绍特殊情况）

```
public void go() {
    JFrame frame = new JFrame();

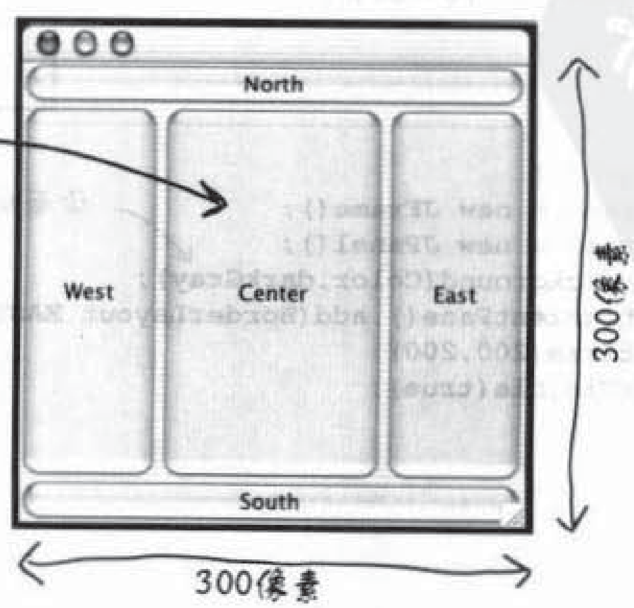
    JButton east = new JButton("East");
    JButton west = new JButton("West");
    JButton north = new JButton("North");
    JButton south = new JButton("South");
    JButton center = new JButton("Center");

    frame.getContentPane().add(BorderLayout.EAST, east);
    frame.getContentPane().add(BorderLayout.WEST, west);
    frame.getContentPane().add(BorderLayout.NORTH, north);
    frame.getContentPane().add(BorderLayout.SOUTH, south);
    frame.getContentPane().add(BorderLayout.CENTER, center);

    frame.setSize(300,300);
    frame.setVisible(true);
}
```

中央的组件大小要看扣除周围之后还剩下些什么

东西会取得预设的宽度南北会取得预设的高度



南北会先占位，所以东西的高度还要扣除南北的高度



FlowLayout布局组件的流向：依次从左至右、从上至下

把面板加入东区：

JPanel的布局管理器的默认布局是FlowLayout布局。当我们把面板加到框架时，面板的大小与位置还是受BorderLayout布局的管理。但面板内部的组件（通过panel.add(someComponent)加入）是由面板的FlowLayout布局来管理的。我们先把空的面板放到框架的东区，下一页会再加组件到面板上。

面板还没有东西在上面，所以不会要求太多的区域



```
import javax.swing.*;
import java.awt.*;

public class Panel1 {

    public static void main (String[] args) {
       Panel1 gui = new Panel1();
       gui.go();
   }

   public void go() {
       JFrame frame = new JFrame();
       JPanel panel = new JPanel();
       panel.setBackground(Color.darkGray);
       frame.getContentPane().add(BorderLayout.EAST, panel);
       frame.setSize(200,200);
       frame.setVisible(true);
   }
}
```

让面板变成深灰色以便观察

把按钮加到面板上

```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    JButton button = new JButton("shock me");

    panel.add(button);
    frame.getContentPane().add(BorderLayout.EAST, panel);

    frame.setSize(250,200);
    frame.setVisible(true);
}
```

把按钮加到面板上面





如果加两个按钮到面板上?

```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");

    panel.add(button);
    panel.add(buttonTwo);

    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

创建两个按钮

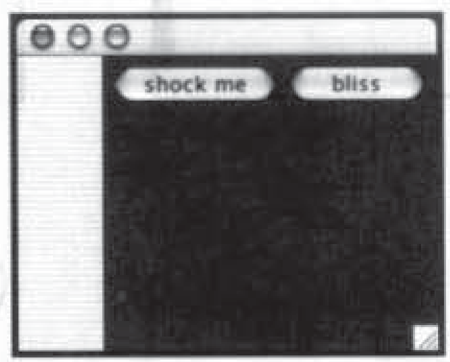
加到面板上

想要的效果：



垂直排列

实际的效果：



横排

注意到字少的宽度比较小，顺序布局会让按钮取得刚好所需的大小

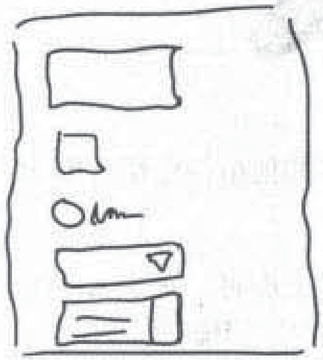
Sharpen your pencil

如果上面的程序改成下面这样，GUI会长得像什么样子?

```
JButton button = new JButton("shock me");
JButton buttonTwo = new JButton("bliss");
JButton buttonThree = new JButton("huh?");
panel.add(button);
panel.add(buttonTwo);
panel.add(buttonThree);
```



把你认为的结果画出来!



BoxLayout布局是救星！

就算够宽它还是会垂直排列

不像FlowLayout布局，就算水平宽度足以容纳组件，它还是会用新的行来排列组件

所以你现在必须把面板的布局管理器从默认的FlowLayout布局改成BoxLayout布局

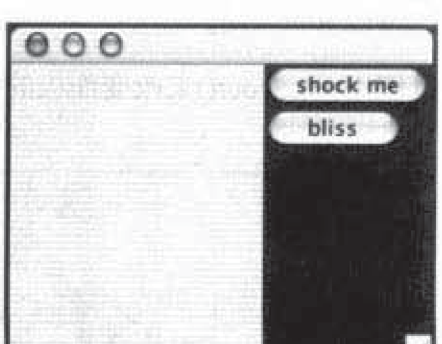
```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");
    panel.add(button);
    panel.add(buttonTwo);
    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

把布局管理器换掉

它的构造函数需要知道要管理哪个组件以及使用哪个轴



面板又变窄了，因为不需要水平地塞入组件，因此只要够放最宽的那一个就可以

there are no Dumb Questions

问：框架为什么不能像面板那样直接地加上组件？

答：JFrame会这么特殊是因为它是让事物显示在画面上的接点。因为Swing的组件纯粹由Java构成，JFrame必须要连接到底层的操作系统以便来存取显示装置。我们可以把面板想做事安置在JFrame上的100%纯Java层。或者把JFrame想做是支撑面板的框架。你甚至可以用自定义的JPanel来换掉框架的面板：

```
myFrame.setContentPane(myPanel);
```

问：我能够换掉框架的布局管理器吗？如果我想让框架用 顺序替换边界呢？

答：最简单的方法是创建一个面板，让此面板成为框架 的content pane，使得GUI以你想要的方式运行。

问：如果想要有不同的理想大小应该怎么办？组件是否有setSize()方法？

答：是有setSize()，但布局管理器会把它忽略掉。组件理想的大小与你想要的大小是有差距的。理想的大小是根据组件确实所需的大小来计算的（组件自行计算）。布局管理器会调用组件的getPreferredSize()方法，而此方法并不会考虑你之前对setSize()的调用。

问：能不能直接定位？能不能关掉布局管理器？

答：你可以调用setLayout(null)直接设定画面位置和大小。但使用布局管理器还是比较好的方式。

要点

- 布局管理器会控制嵌套在其他组件中组件的大小和位置。
- 当某个组件加到背景组件上面时，被加入的组件是由背景组件的布局管理器管理的。
- 布局管理器在做决定之前会询问组件的理想大小，并根据策略来决定采用哪些数据。
- BorderLayout布局可以让你把组件加到五个区域上。你必须以下列语法来指定区域：

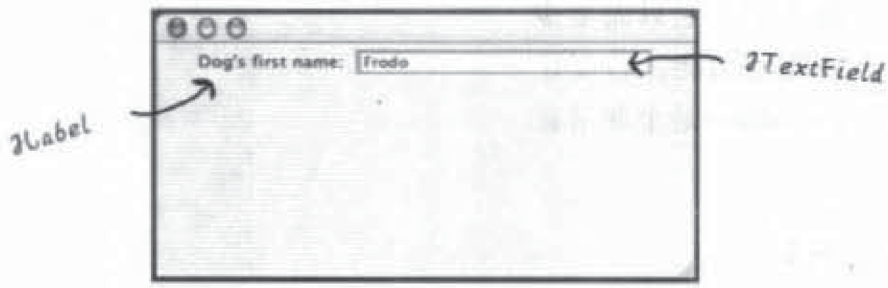
  ```
  add(BorderLayout.EAST, panel);
  ```

- BorderLayout布局上的南北区域使用组件的理想高度而不管宽度，东西区域刚好相反，中间区域只能使用剩下的空间。
- pack()方法会使window的大小符合内含组件的大小。
- FlowLayout布局会由左至右、由上至下依加入的顺序来安置组件，若宽度超过时就会换行。
- FlowLayout布局会让组件在长宽上都使用理想的尺寸大小。
- BoxLayout布局让你可以垂直地排列组件，如同FlowLayout布局一样，它会让组件在长宽上都使用理想的尺寸大小。
- 框架默认时使用BoxLayout布局，面板默认使用FlowLayout布局。
- 可以调用setLayout()来改变面板的布局管理器

操作Swing组件

你已经看过布局管理器的基本说明，因此现在就让我们来看一下几个最常用的组件：text field、可滚动的text area、checkbox以及list。我们不打算把整个API 都拿出来说一遍，只讲几个重点。

JTextField



构造函数

20代表20字宽而不是像素

```
JTextField field = new JTextField(20);
JTextField field = new JTextField("Your name");
```

如何使用

① 取得文本内容。

```
System.out.println(field.getText());
```

② 设定内容。

```
field.setText("whatever");
field.setText("");
```

清空字段

③ 取得用户输入完毕按下return或enter键的事件。

如果想要知道用户的每个按键操作也可以注册按键的事件

```
field.addActionListener(myActionListener);
```

④ 选取文本字段的内容。

```
field.selectAll();
```

⑤ 把GUI目前焦点拉回到文本字段以便让用户进行输入操作。

```
field.requestFocus();
```

JTextArea



不像JTextField，JTextArea可以有超过一行以上的文字。它只需要少许的设定就可制作出来，因为这样没有做滚动条或换行功能。若要让JTextArea滚动，就必须要把它粘在ScrollPane上。ScrollPane是个非常喜欢滚动的对象，并也会考虑文本区域的滚动需求。

构造函数

代表10行高

20字宽

```
JTextArea text = new JTextArea(10,20);
```

如何使用

1. 只有垂直的滚动条。

```
JScrollPane scroller = new JScrollPane(text);
text.setLineWrap(true);

scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

panel.add(scroller);
```

将text赋值给新创建的JScrollPane

启动自动换行

指定只使用垂直滚动条

这很重要，加入的是带有文本域的滚动条而不是文本域！

2. 替换掉文字内容。

```
text.setText("Not all who are lost are wandering");
```

3. 加入文字。

```
text.append("button clicked");
```

4. 选取内容。

```
text.selectAll();
```

5. 把GUI目前焦点拉回到文本字段以便让用户进行输入操作。

```
text.requestFocus();
```

JTextArea范例




```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class TextArea1 implements ActionListener {

    JTextArea text;

    public static void main (String[] args) {
       TextArea1 gui = new TextArea1();
       gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      JButton button = new JButton("Just Click It");
      button.addActionListener(this);
      text = new JTextArea(10,20);
      text.setLineWrap(true);

      JScrollPane scroller = new JScrollPane(text);
      scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
      scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

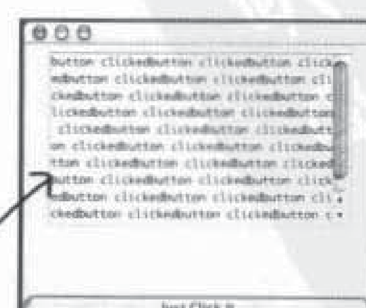
      panel.add(scroller);

      frame.getContentPane().add(BorderLayout.CENTER, panel);
      frame.getContentPane().add(BorderLayout.SOUTH, button);

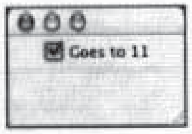
      frame.setSize(350,300);
      frame.setVisible(true);
   }

   public void actionPerformed(ActionEvent ev) {
      text.append("button clicked \n ");
   }
}
```

在按下按钮时插入一个换行字符，不然的话都会粘在一起



JCheckBox



构造函数

```
JCheckBox check = new JCheckBox("Goes to 11");
```

如何使用

① 监听 item 的事件（被选取或变成非选取）。

```
check.addItemListener(this);
```

② 处理事件（判别是否被选取）。

```
public void itemStateChanged(ItemEvent ev) {
    String onOrOff = "off";
    if (check.isSelected()) onOrOff = "on";
    System.out.println("Check box is " + onOrOff);
}
```

③ 用程序来选取或不选取。

```
check.setSelected(true);
check.setSelected(false);
```

there are no Dumb Questions

问：布局管理器产生的问题是不是比所解决的问题更多？如果我得处理这么多的问题，那我得考虑是否直接设定位置和大小算了。

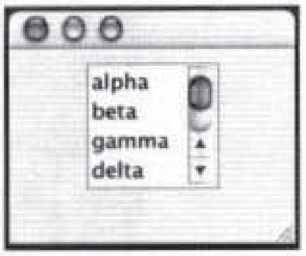
答： 以布局管理器取得完全符合想法的布局是个很大的挑战。但要考虑到它还帮你做了哪些事情。就算是计算组件要出现在画面的什么位置也是很复杂的工作。举例来说，布局管理器能够防止组件互相覆盖。也就是说，它知道如何管理组件（以及框架）的间距。你当然可以自己做到这样的功能，但如果组件多到一种程度你还会想要手动的来调整吗？这只有对Java虚拟机有好处而已！

为什么？因为组件在不同平台上长得不太一样，像是在某平台上刚好并排按钮边缘在另外一个平台上可能就重叠了。

这还不是最麻烦的，只要想象用户调整window大小时会发生什么事。你应该会很庆幸有机会不用自己写这么多与真正商业逻辑无关的程序代码吧？

PDG

JList



JList的构造函数需要一个任意类型的数组。不一定要是String，但会用String来表示项目

构造函数

```
String [] listEntries = {"alpha", "beta", "gamma", "delta",
                         "epsilon", "zeta", "eta", "theta "};

list = new JList(listEntries);
```

如何使用

与JTextArea相同，要放在JScrollPane上面

① 让它显示垂直的滚动条。

```
JScrollPane scroller = new JScrollPane(list);
scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

panel.add(scroller);
```

② 设定显示的行数。

```
list.setVisibleRowCount(4);
```

③ 限制用户只能选取一个项目。

```
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

④ 对选择事件做注册。

```
list.addListSelectionListener(this);
```

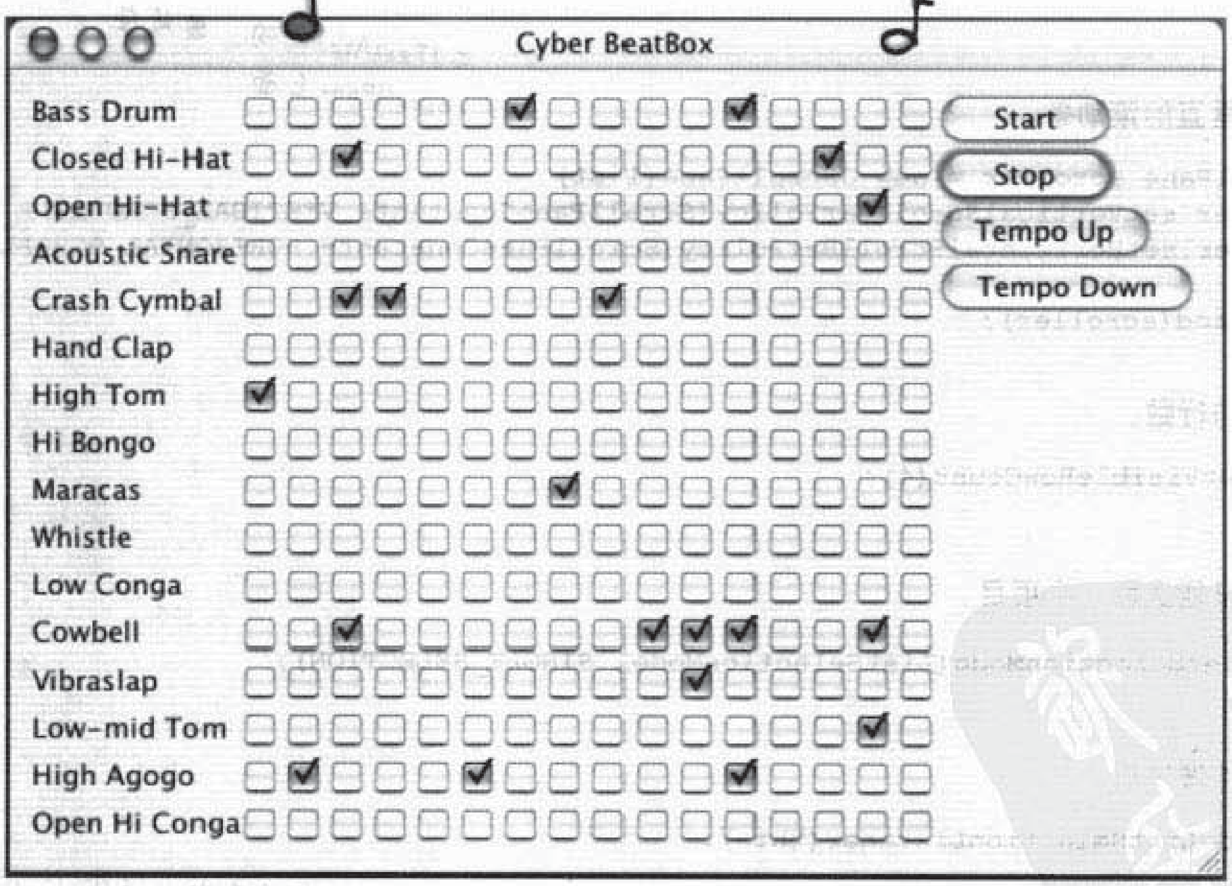
⑤ 处理事件（判断选了哪个项目）。

如果没有加上这个 if 测试，你会得到两次的事件

```
public void valueChanged(ListSelectionEvent lse) {
    if( !lse.getValueIsAdjusting()) {
      String selection = (String) list.getSelectedValue();
      System.out.println(selection);
    }
}
```

这会返回一个Object，不一定是个String

程序料理



这一段是选择性的内容，我们会来创建出完整版本的BeatBox。我们在讨论对象存储的章节会学到如何存储与打开节奏设定。在讨论网络的章节中我们会把BeatBox改成聊天室的客户端程序。

创建BeatBox

接下来有这一版BeatBox的程序行表。其中带有启动、停止和改变节奏的按钮。这是个完整的程序行表，并有注释。以下是这个程序的概要。

1. **创建出带有256个复选框的GUI。初始的时候这些复选框都是未勾选的，乐器的名称用到16个JLabel，还有4个按钮。**

2. **对上面的4个按钮注册ActionListener。我们无需个别的监听复选框，因为我们不会试着动态地（在复选框被点选时）马上改变发声的样式。相反，我们会等到用户按下start按钮之后才会检查这256个复选框的状态并制作出MIDI的track。**

3. **设定MIDI系统（之前就已经做过了），这包括取得Sequencer、创建Sequence以及track。我们会用到Java 5.0之后才有的setLoopCount()这个sequencer的方法。它能让你指定重复播放的次数。我们也会用到节奏因子（tempo factor）来调整节奏的速度，并维持重复时的节奏。**

4. **当用户按下start时，启动真正的操作。此按钮的事件处理程序会调用butildTrackAndStart()方法。在该方法中，我们会逐个（一次一行）取得256个复选框的状态，然后使用这些信息来创建MIDI的track（使用之前编写的makeEvent()）。一旦track完成之后，我们会启动sequencer来持续播放直到用户按下stop为止。**

```
import java.awt.*;
import javax.swing.*;
import javax.sound.midi.*;
import java.util.*;
import java.awt.event.*;


public class BeatBox {

    JPanel mainPanel;
    ArrayList<JCheckBox> checkboxList;
    Sequencer sequencer;
    Sequence sequence;
    Track track;
    JFrame theFrame;

    String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat",
       "Open Hi-Hat","Acoustic Snare", "Crash Cymbal", "Hand Clap",
       "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga",
       "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo",
       "Open Hi Conga"};
    int[] instruments = {35,42,46,38,49,39,50,60,70,72,64,56,58,47,67,63};


    public static void main (String[] args) {
        new BeatBox2().buildGUI();
    }

    public void buildGUI() {
        theFrame = new JFrame("Cyber BeatBox");
        theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BorderLayout layout = new BorderLayout();
        JPanel background = new JPanel(layout);
        background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

        checkboxList = new ArrayList<JCheckBox>();
        Box buttonBox = new Box(BoxLayout.Y_AXIS);

        JButton start = new JButton("Start");
        start.addActionListener(new MyStartListener());
        buttonBox.add(start);

        JButton stop = new JButton("Stop");
        stop.addActionListener(new MyStopListener());
        buttonBox.add(stop);

        JButton upTempo = new JButton("Tempo Up");
        upTempo.addActionListener(new MyUpTempoListener());
        buttonBox.add(upTempo);

        JButton downTempo = new JButton("Tempo Down");
```

把checkbox储存在ArrayList中

乐器的名称，以String的array维护

实际的乐器关键字，例如说35是bass，42是Closed Hi-Hat

设定面板上摆设组件时的空白边缘

一般的GUI程序代码

```
    downTempo.addActionListener(new MyDownTempoListener());
    buttonBox.add(downTempo);

    Box nameBox = new Box(BoxLayout.Y_AXIS);
    for (int i = 0; i < 16; i++) {
       nameBox.add(new Label(instrumentNames[i]));
    }

    background.add(BorderLayout.EAST, buttonBox);
    background.add(BorderLayout.WEST, nameBox);

    theFrame.getContentPane().add(background);

    GridLayout grid = new GridLayout(16,16);
    grid.setVgap(1);
    grid.setHgap(2);
    mainPanel = new JPanel(grid);
    background.add(BorderLayout.CENTER, mainPanel);

    for (int i = 0; i < 256; i++) {
        JCheckBox c = new JCheckBox();
        c.setSelected(false);
        checkboxList.add(c);
        mainPanel.add(c);
    } // 循环结束

    setUpMidi();

    theFrame.setBounds(50,50,300,300);
    theFrame.pack();
    theFrame.setVisible(true);
} // 关闭方法


public void setUpMidi() {
  try {
    sequencer = MidiSystem.getSequencer();
    sequencer.open();
    sequence = new Sequence(Sequence.PPQ,4);
    track = sequence.createTrack();
    sequencer.setTempoInBPM(120);

  } catch(Exception e) {e.printStackTrace();}
} // 关闭方法
```

也是一般的程序代码

创建checkbox组，设定成未勾选的为false并加到ArrayList和面板上

一般的MIDI设置程序代码

重点在这里！此处会将复选框状态转换为MIDI事件并加到track上

```
public void buildTrackAndStart() {
  int[] trackList = null;            // 创建出16个元素的数组来存储一项乐器的值。如果该节应该要演奏，其值会是关键字值，否则值为零

  sequence.deleteTrack(track);       // 清除掉旧的track做一个新的
  track = sequence.createTrack();

    for (int i = 0; i < 16; i++) {   // 对每个乐器都执行一次
      trackList = new int[16];

      int key = instruments[i];      // 设定代表乐器的关键字

      for (int j = 0; j < 16; j++ ) {   // 对每一拍执行一次

          JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
          if ( jc.isSelected()) {
             trackList[j] = key;     // 如果有勾选，将关键字值放到数组的该位置上，不然的话就补零
          } else {
             trackList[j] = 0;
          }
       } // 关闭内部循环

       makeTracks(trackList);        // 创建此乐器的事件并加到track上
       track.add(makeEvent(176,1,127,0,16));
    } // 关闭外部循环

    track.add(makeEvent(192,9,1,0,15));   // 确保第16拍有事件，否则beatbox不会重复播放
    try {

       sequencer.setSequence(sequence);
        sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);   // 指定无穷的重复次数
        sequencer.start();
        sequencer.setTempoInBPM(120);
    } catch(Exception e) {e.printStackTrace();}   // 开始播放！
} // 关闭 buildTrackAndStart方法

public class MyStartListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        buildTrackAndStart();
     }
} // 关闭内部类
```

第一个内部类，按钮的监听者

```
public class MyStopListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        sequencer.stop();
    }
} // 关闭内部类

public class MyUpTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
      float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * 1.03));
    }
 } // 关闭内部类

 public class MyDownTempoListener implements ActionListener {
     public void actionPerformed(ActionEvent a) {
      float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * .97));
    }
} // 关闭内部类
```

另一个内部类，也是按钮的监听者

节奏因子，预设为1.0，每次调整3%

创建某项乐器的所有事件

```
public void makeTracks(int[] list) {

   for (int i = 0; i < 16; i++) {
      int key = list[i];

      if (key != 0) {
         track.add(makeEvent(144,9,key, 100, i));
         track.add(makeEvent(128,9,key, 100, i+1));
      }
   }
}
```

创建NOTE ON和NOTE OFF事件并加入到track上

```
public  MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
        ShortMessage a = new ShortMessage();
        a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);

    } catch(Exception e) {e.printStackTrace(); }
    return event;
}

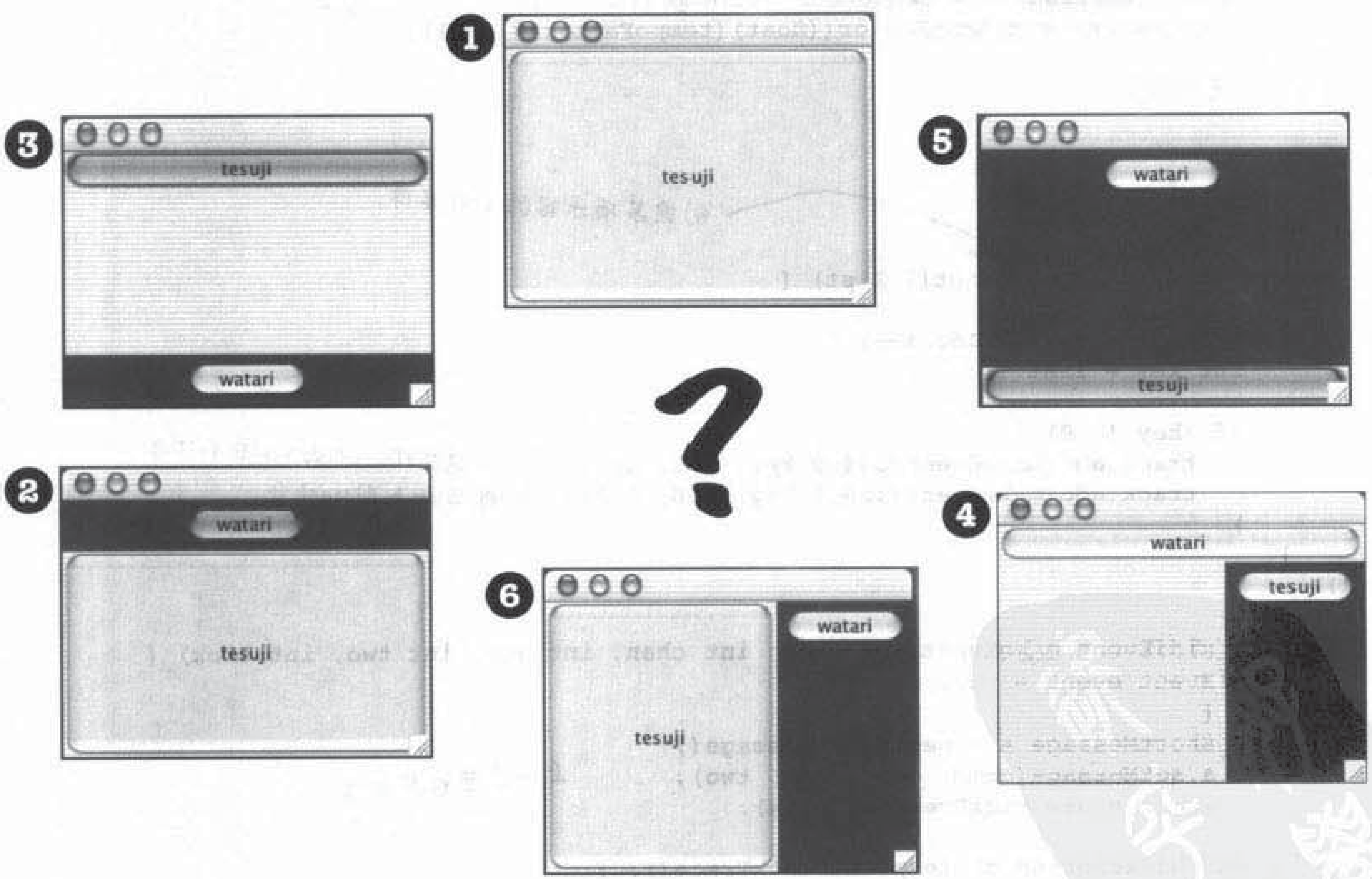
} // 关闭类
```

上一章已经用过了



哪一个程序用了哪一个layout?

下面的6个画面中有5个是由下一页的程序段所产生的。找出哪个画面是由哪个程序段落所产生的。



程序段落

D

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

B

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

C

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

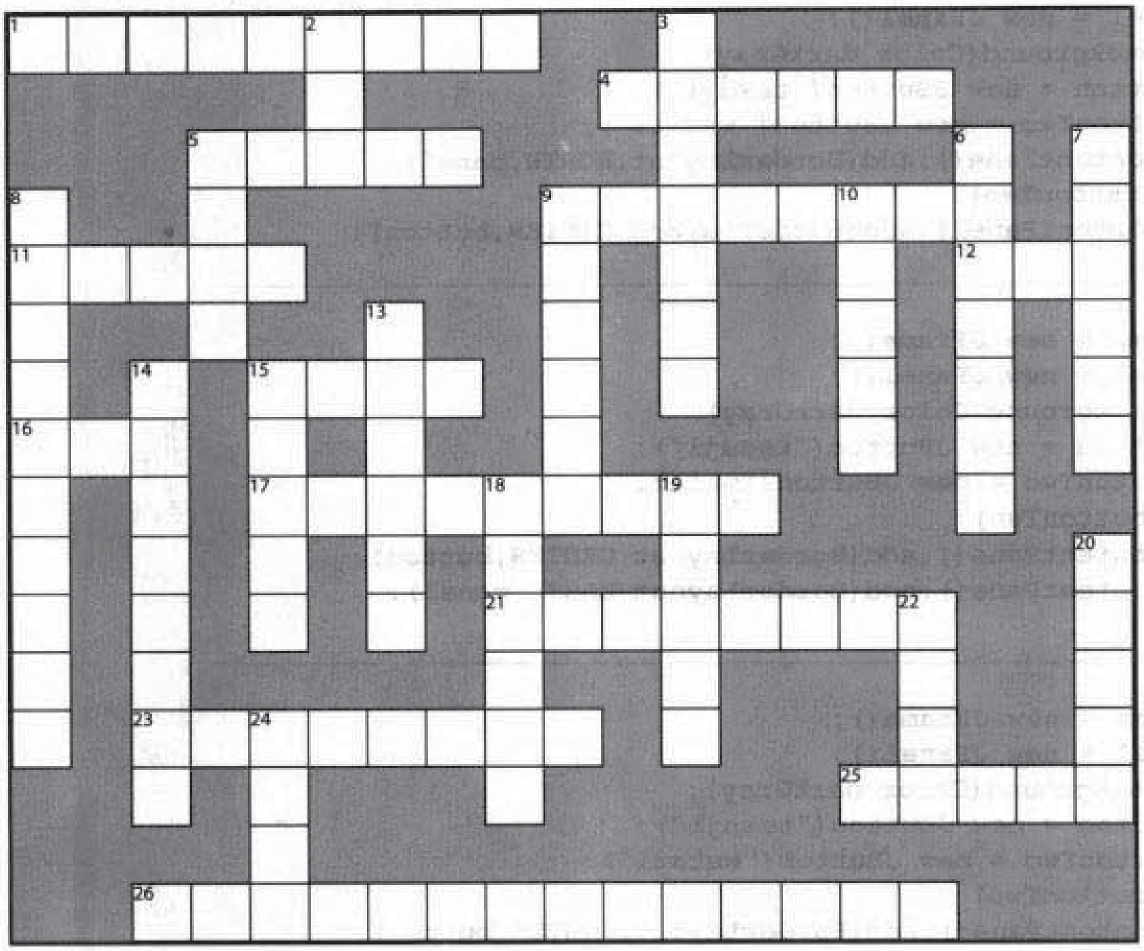
A

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH,buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

E

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH,button);
```

GUI-Cross 7.0



你行吗？

横排提示：

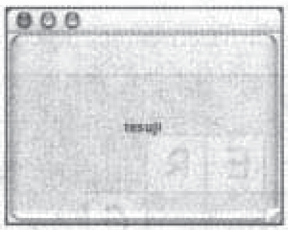
1. Artist's sandbox
4. Border's catchall
5. Java look
9. Generic waiter
11. A happening
12. Apply a widget
15. JPanel's default
16. Polymorphic test
17. Shake it baby
21. Lots to say
23. Choose many
25. Button's pal
26. Home of actionPerformed

竖排提示：

2. Swing's dad
3. Frame's purview
5. Help's home
6. More fun than text
7. Component slang
8. Romulin command
9. Arrange
10. Border's top
13. Manager's rules
14. Source's behavior
15. Border by default
18. User's behavior
19. Inner's squeeze
20. Backstage widget
22. Mac look
24. Border's right

练习解答

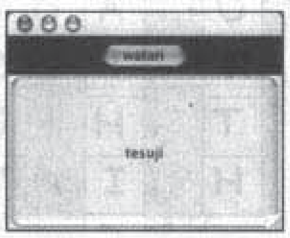
1



C

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

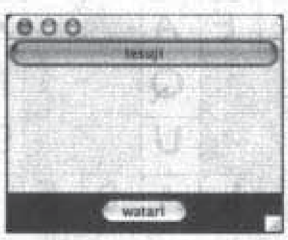
2



D

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
```

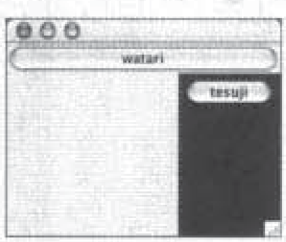
3



E

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH,panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH,button);
```

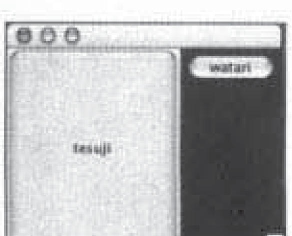
4



A

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH,buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

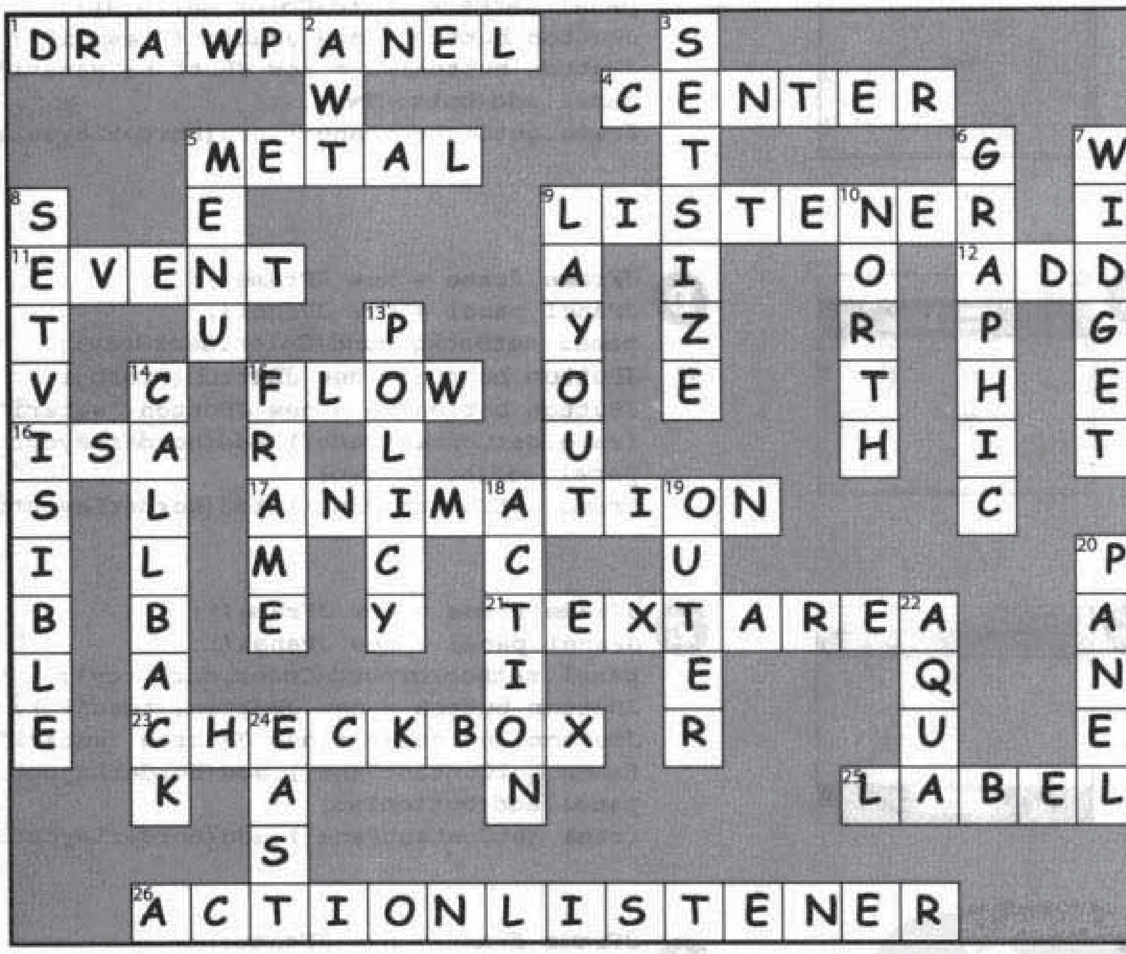
6



B

```
JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER,button);
frame.getContentPane().add(BorderLayout.EAST, panel);
```

GUI-Cross 7.0 解答



14 序列化和文件的输入/输出

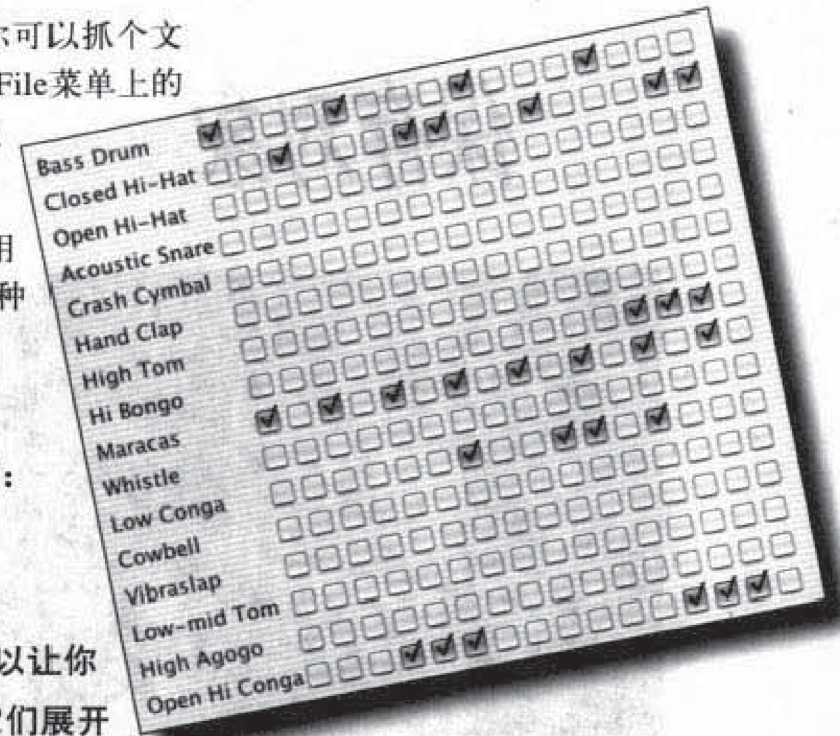
保存对象



对象可以被序列化也可以展开。对象有状态和行为两种属性。行为存在于类中，而状态存在于个别的对象中。所以需要存储对象状态的时候会发生什么事？如果你正在编写游戏，就得有存储和恢复游戏的功能。如果你编写的是创建图表的程序，也必须要有储存/打开的功能。如果程序需要储存状态，你可以来硬的，对每一个对象，逐个地把每项变量的值写到特定格式的文件中。或者，你也可以用面向对象的方式来做——只要把对象本身给冻干/辗平/保存/脱水，并加以重组/展开/恢复/泡开成原状。但有时这还得来硬的，特别是在程序所储存的文件需要给某些非Java的应用程序所读取时，所以这一章会讨论这两种方式。

抓住节奏

你已经奏出完美的乐章，现在会想把它储存起来。你可以抓个文房四宝把它记下来，但也可以按下储存按钮（或按下File菜单上的Save）。然后你帮文件命名，并希望这个文件不会让屏幕变成蓝色的画面。



储存状态的选择有很多种，这可能要看你会如何使用储存下来的状态而决定。我们会在这一章讨论下面两种选项：

如果只有自己写的Java程序会用到这些数据：

① 用序列化（serialization）。

将被序列化的对象写到文件中。然后就可以让你的程序去文件中读取序列化的对象并把它们展开回到活生生的状态。

如果数据需要被其他程序引用：

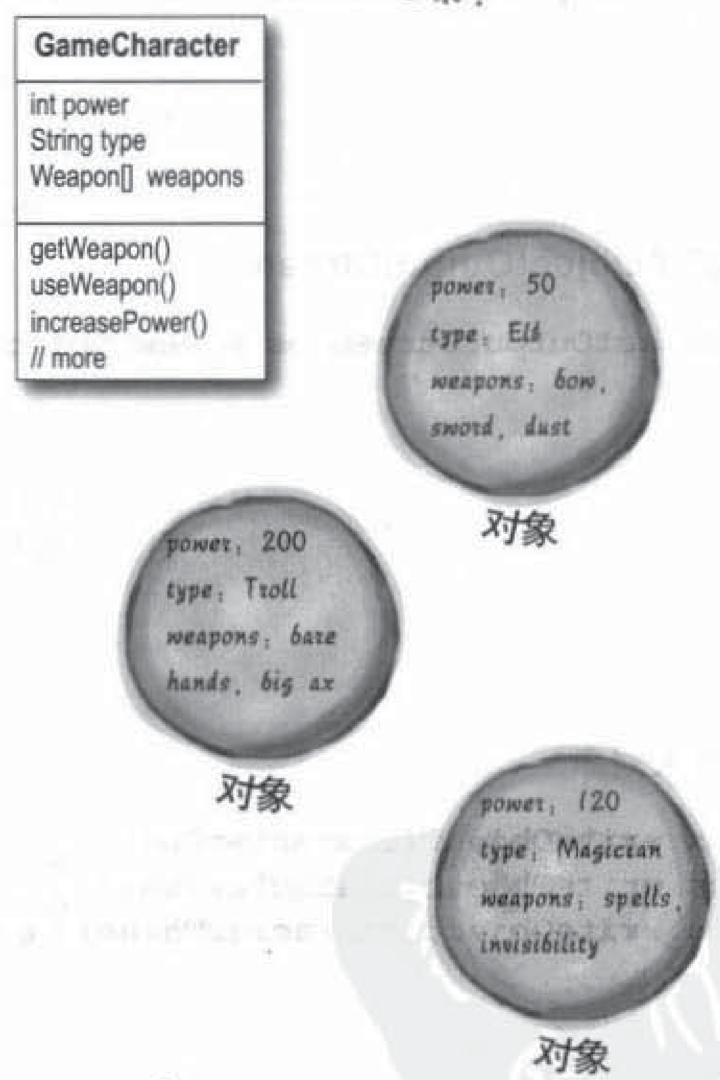
② **写一个纯文本文件。用其他程序可以解析的特殊字符写到文件中。例如写成用tab字符来分隔的档案以便让电子表格或数据库应用程序能够应用。**

当然还有其他的选择。你可以将数据存进任何格式中。举例来说，你可以把数据用字节而不是字符来写入，或者你也可以将数据写成Java的primitive主数据类型，有一些方法可以提供int、long、boolean等的写入功能。但不管用什么方法，基本所需的输入/输出技巧都一样：把数据写到某处，这可能是个磁盘上的文件，或者是来自网络上的串流。读取数据的方向则刚好相反。当然此处所讨论的部分不涉及使用数据库的情况。

存储状态

假设你有个程序，是个幻想冒险游戏，要过很多关才能完成。在游戏进行的过程中，游戏的人物会累积经验值、宝物、体力等。你不会想让游戏每次重新启动时都得要从头来过——这样根本没人玩。因此你需要一种方法来保存人物的状态，并且在重新开启时能够将状态回复到上次存储时的原状。因为你是程序员，所以你的工作是要让存储与恢复尽可能的简单容易。

假设有精灵、矮人、和魔法师等3种人物要记录：



① 选项一

把3种序列化的人物对象写入文件中。

创建一个文件，让序列化的3种对象写到此文件中。这文件在你以文本文件形式阅读时是无意义的：

"isr GameCharacter "%gê8MÛIpowerLjava/lang/String;[weaponst[Ljava/lang/String;xp2tlfur[Ljava.lang.String;"VÁ È{Gxptbowtswordtdustsq~» tTrolluq~tbare handstbig axsq~xtMagicianuq~tspellstinvisibility

② 选项二

写入纯文本文件

创建文件，写入3行文字，每个人物一行，以逗点来分开属性：

50,Elf,bow, sword,dust
200,Troll,bare hands,big ax
120,Magician,spells,invisibility

序列化的文件是很难让一般人阅读的，但它比纯文本文件更容易让程序恢复这3种人物的状态，也比较安全，因为一般人不会知道要如何动手脚改数据

将序列化对象写入文件

下面是将对象序列化（存储）的方法步骤。不用硬记下来，后面的内容会有详细的说明。

如果文件不存在，它会自动被创建出来

1 创建出FileOutputStream

```
FileOutputStream fileStream = new FileOutputStream("MyGame.ser");
```

创建存取文件的FileOutputStream对象

2 创建ObjectOutputStream

```
ObjectOutputStream os = new ObjectOutputStream(fileStream);
```

它能让你写入对象，但无法直接地连接文件，所以需要参数的指引

3 写入对象

```
os.writeObject(characterOne);
os.writeObject(characterTwo);
os.writeObject(characterThree);
```

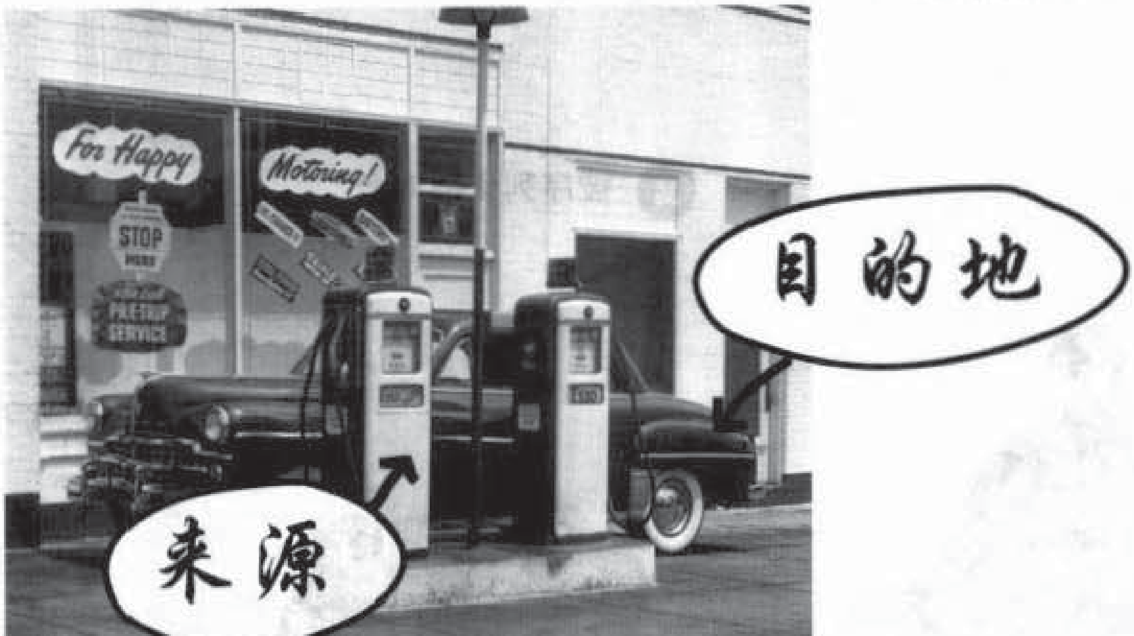
将变量所引用的对象序列化并写入MyGame.ser这个文件

4 关闭ObjectOutputStream

```
os.close();
```

关闭所关联的输出串流

数据在串流中移动



将串流（stream）连接起来代表来源与目的地（文件或网络端口）的连接。串流必须要连接到某处才能算是个串流。

Java的输入/输出API带有连接类型的串流，它代表来源与目的地之间的连接，连接串流将串流与其他串流连接起来。

一般来说，串流要两两连接才能作出有意义的事情——其中一个表示连接，另一个则是要被调用方法的。为何要两个？因为连接的串流通常都是很低层的。以FileOutputStream为例，它有可以写入字节的方法。但我们通常不会直接写字节，而是以对象层次的观点来写入，所以需要高层的连接串流。

那又为何不以单一的串流来执行呢？这就要考虑到良好的面向对象设计了。每个类只要做好一件事。FileOutputStream把字节写入文件。ObjectOutputStream把对象转换成可以写入串流的数据。当我们调用ObjectOutputStream的writeObject时，对象会被打成串流送到FileOutputStream来写入文件。

这样就可以通过不同的组合来达到最大的适应性！如果只有一种串流类的话，你只好祈祷API的设计人已经想好所有可能的排列组合。但通过链接的方式，你可以自由地安排串流的组合与去向。



对象被序列化的时候发生了什么事？

❶ 在堆上的对象

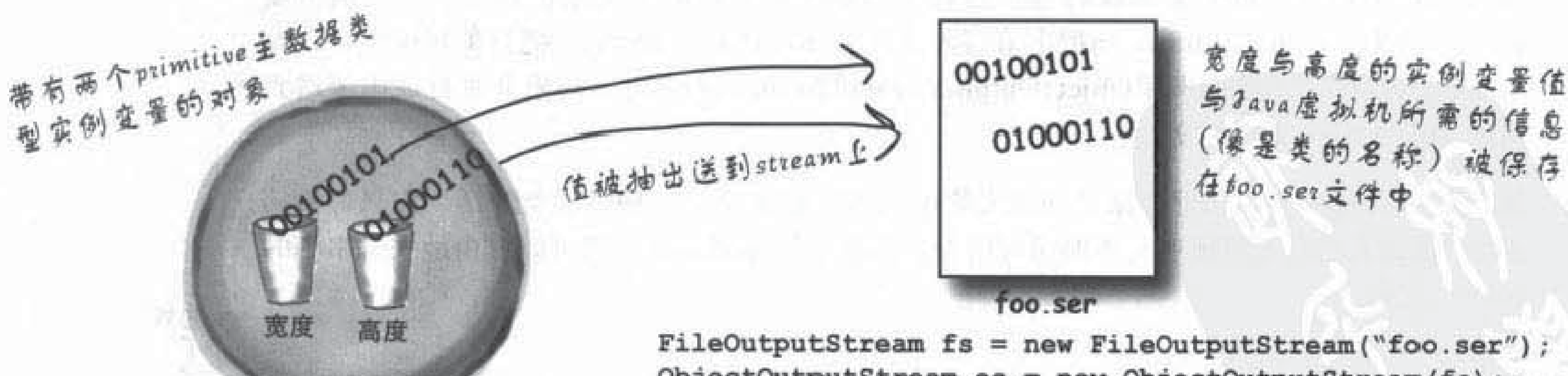


在堆上的对象有状态——实例变量的值。这些值让同一类的不同实例有不同的意义。

❷ 被序列化的对象



序列化的对象保存了实例变量的值，因此之后可以在堆上带回一模一样的实例。




```
FileOutputStream fs = new FileOutputStream("foo.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);
os.writeObject(myFoo);
```

```
Foo myFoo = new Foo();
myFoo.setWidth(37);
myFoo.setHeight(70);
```

创建出FileOutputStream链接到ObjectOutputStream以让它写入对象

对象的状态是什么？有什么需要保存？

事情开始有趣了。存储primitive主数据类型值37和70是很简单的。但如果对象有引用到其他对象的实例变量时要怎么办？如果这些对象还带有其他对象又该如何？

想想看吧。对象基本上有哪些部分是独特的？有哪些东西需要被带回来才能让对象回到和存储时完全相同的状态？当然它会有不同的内存位置，但这无关紧要。我们在乎的是堆上是否有与存储时一模一样的对象状态。

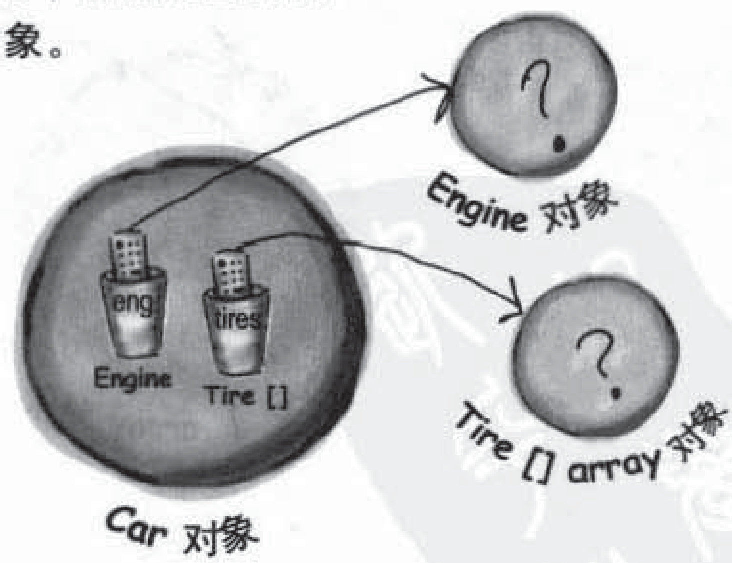
Brain Barbell

要怎样才能让Car对象能够存储成可以恢复原始状态的形式？

考虑一下有哪些东西是必要保存的。

如果引擎对象还有对汽缸的引用时该如何？而Tire数组对象里面会有什么东西？

Car对象有两个实例变量引用到其他的对象。



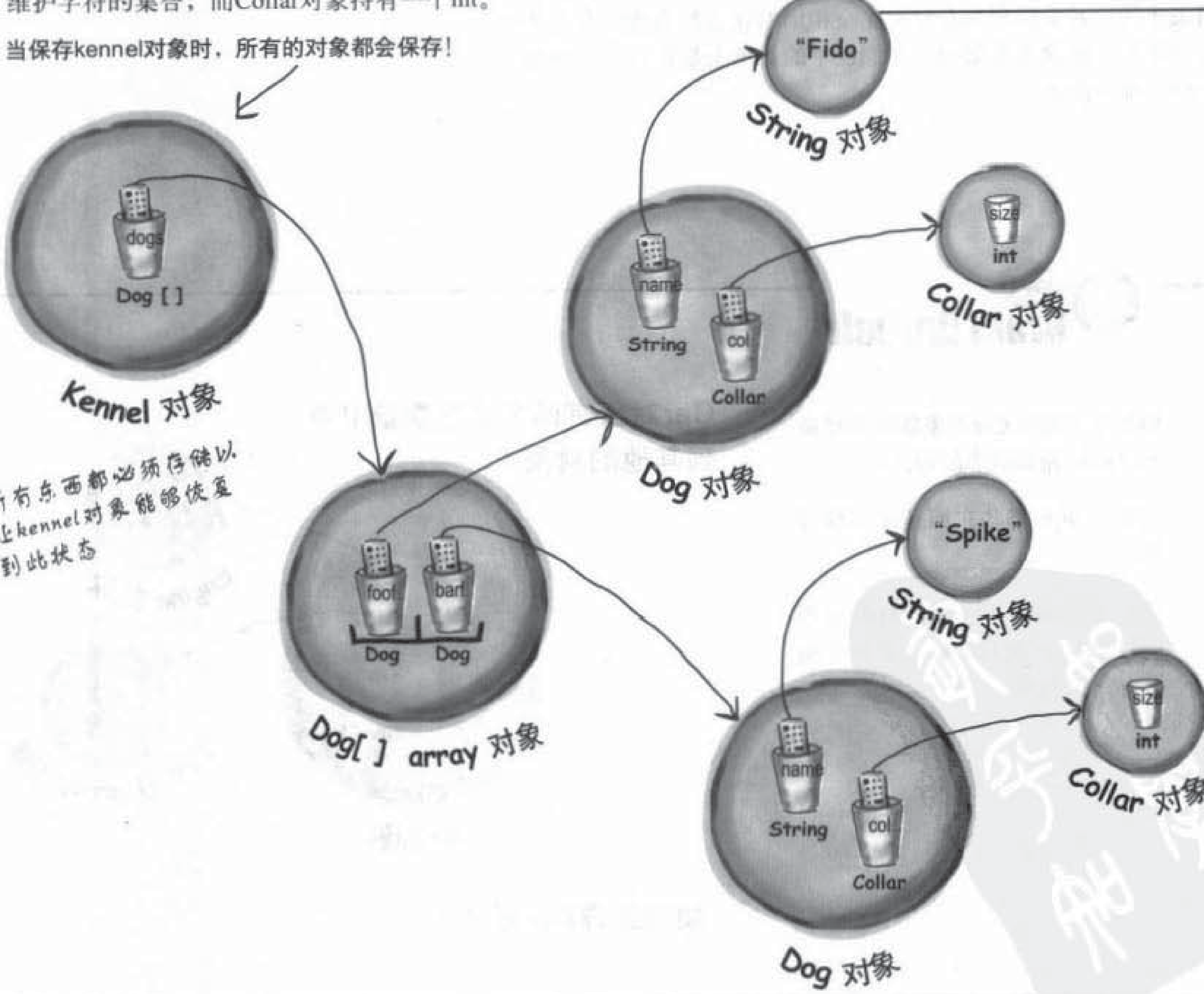
如何保存Car对象？

当对象被序列化时，被该对象引用的实例变量也会被序列化。且所有被引用的对象也会被序列化……最棒的是，这些操作都是自动进行的！

序列化程序会将对象版图上的所有东西存储起来。被对象的实例变量所引用的所有对象都会被序列化。

Kennel对象带有对Dog数组对象的引用。Dog[]中有两个Dog对象的引用。每个Dog对象带有String和Collar对象的引用。String对象维护字符的集合，而Collar对象持有一个int。

当保存kennel对象时，所有的对象都会保存！



所有东西都必须存储以让kennel对象能够恢复到此状态

如果要让类能够被序列化，就实现Serializable

Serializable接口又被称为marker或tag类的标记用接口，因为此接口并没有任何方法需要实现的。它的唯一目的就是声明有实现它的类是可以被序列化的。也就是说，此类型的对象可以通过序列化的机制来存储。如果某类是可序列化的，则它的子类也自动地可以序列化（接口的本意就是如此）。

```
objectOutputStream.writeObject(myBox);
```

任何放在此处的对象都必须要实现序列化，否则在执行期一定会出问题

```
import java.io.*;

public class Box implements Serializable {

    private int width;
    private int height;

    public void setWidth(int w) {
        width = w;
    }

    public void setHeight(int h) {
        height = h;
    }

public static void main (String[] args) {

        Box myBox = new Box();
        myBox.setWidth(50);
        myBox.setHeight(20);

        try {
            FileOutputStream fs = new FileOutputStream("foo.ser");
            ObjectOutputStream os = new ObjectOutputStream(fs);
            os.writeObject(myBox);
            os.close();
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

必须要import它

没有方法需要被实现，只是用来告诉Java虚拟机它可以被序列化

这些值会被保存

有可能会抛出异常

如果不存在就会被创建出来

设定链接

序列化是全有或全无的

你能想象只有部分状态被正确保存的下场吗？



整个对象版图都必须正确地序列化，不然就得全部失败。

如果Duck对象不能序列化，Pond对象就不能被序列化。

```
import java.io.*;

public class Pond implements Serializable {    // Pond对象可被序列化

    private Duck duck = new Duck();    // 它有个Duck实例变量

    public static void main (String[] args) {
       Pond myPond = new Pond();
       try {
         FileOutputStream fs = new FileOutputStream("Pond.ser");
ObjectOutputStream os = new ObjectOutputStream(fs);

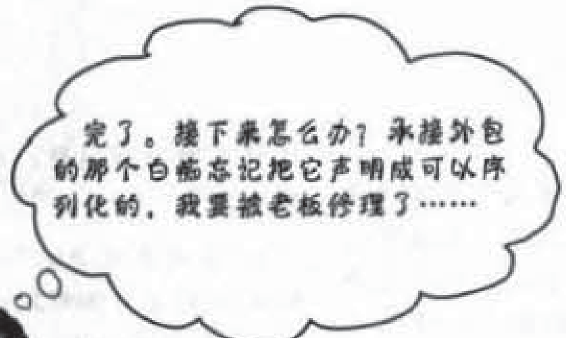
         os.writeObject(myPond);    // 将myPond序列化的同时Duck也会被序列化
         os.close();

      } catch(Exception ex) {
          ex.printStackTrace();
      }
   }
}

public class Duck {    // 啊！Duck不能被序列化，因为没有实现序列化！
   // duck 的程序代码
}
```

执行main()的下场：

```
File Edit Window Help Regret
% java Pond
java.io.NotSerializableException: Duck
        at Pond.main(Pond.java:13)
```





如果某实例变量不能或不应该被序列化，就把它标记为transient（瞬时）的

如果你需要序列化程序能够跳过某个实例变量，就把它标记成transient的变量

```
import java.net.*;
class Chat implements Serializable {
    transient String currentID;

    String userName;

    // 还有更多程序代码……
}
```

这会将此变量标记为不需要序列化的

这个变量会被序列化

如果你有无法序列化的变量不能被存储，可以用transient这个关键词把它标记出来，序列化程序会把它跳过。

为什么有些变量不能被序列化？可能是设计者忘记实现Serializable。或者动态数据只可以在执行时求出而不能或不必存储。虽然Java函数库中大部分的类可以被序列化，你还是无法将网络联机之类的东西保存下来。它得要在执行期当场创建才有意义。一旦程序关闭之后，联机本身就不再有用，下次执行时需要重新创建出来。

there are no Dumb Questions

问：如果序列化这么重要，为什么不默认成每个类都有？为什么不让Object实现Serializable以让所有类都自动地成为可序列化的？

答：就算大部分的类都应该实现Serializable，你还是有选择性。且你必须有意识地对各个类做决定。如果它变成默认的，那要如何关掉呢？毕竟界面是用来指示功能性而不是所缺乏的功能。

问：为什么我写过不需要序列化的类？

答：也许是因为安全性的理由让你不想把密码对象存储在文件上。或者某些对象的存储是没有意义的，所以你不会把它实现成可序列化的。这个问题要问施主你自己。

问：为什么我爸爸不是李嘉诚？为什么我长得这么帅却是秃头？

答：这个问题要问施主你自己。

问：如果我使用了一个不能序列化的类，我是否能把它给子类出一个标记为可序列化的类？

答：可以！如果该类是可以被继承（没有被标记为final），你就可以制作出可被序列化的子类。但这又带来另外一个问题：为什么它一开始不是可序列化的？

问：这位同学你问得很好，为什么会有不可序列化类的可序列化子类？

答：这，这，这……首先要看类被还原的时候会发生什么事（稍后会讨论）。简单讲，当对象被还原且它的父类不可序列化时，父类的构造函数会跟创建新的对象一样地执行。如果类没有什么好理由不能被序列化，制作可序列化的子类会是个好方法。

问：我现在才了解，如果你将某个变量标记为trasient的，那就代表说在序列化的过程中该变量会被略过。然后会发生什么事？我们用transient标记变量来解决不能序列化的实例变量问题，但我们不是在回复对象的时候需要该变量吗？序列化的重点不就在于保存对象的状态吗？

答：没错！问题就在这里，但是幸好有解决方法。如果你把某个对象序列化，transient的引用实例变量会以null返回，而不管存储当时它的值是什么。这代表整个对象版图中连接到该特定实例变量的部分不会被存储。这样可能会有问题，所以我们有两个解决方案：

（1）当对象被带回来的时候，重新初始化实例变量回到默认的状态。例如Dog可能会有Collar，但因为Collar无关紧要，所以弄个新的给Dog也不会影响程序逻辑。

（2）如果transient变量的值很重要，例如Collar的颜色是有意义且每个Dog不一样的，你就得需要同时把它的值也保存下来。然后在带回Dog对象时重新创建Collar，再把颜色值设定给Collar。

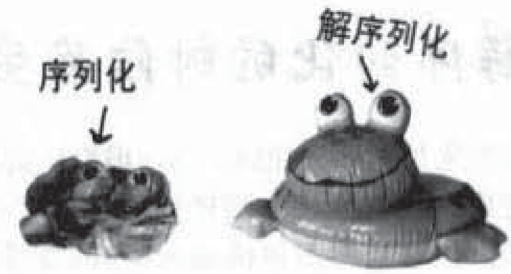
问：如果两个对象都有引用实例变量指向相同的对象会怎样？例如两个Cat都有相同的Owner对象？那Owner会被存储两次吗？

答：好问题！序列化聪明得足以分辨两个对象是否相同。在此情况下只有一个对象会被存储，其他引用会复原成指向该对象。

问：你最喜欢什么课？

答：数学课，因为数学老师常常请假。

解序列化（Deserialization）：还原对象



将对象序列化整件事情的重点在于你可以在事后，在不同的Java虚拟机执行期（甚至不是同一个Java虚拟机），把对象恢复到存储时的状态。解序列化有点像是序列化的反向操作。

1 创建FileInuptStream

```
FileInputStream fileStream = new FileInputStream("MyGame.ser");
```

如果文件不存在就会抛出异常

它知道如何连接文件

2 创建ObjectInputStream

```
ObjectInputStream os = new ObjectInputStream(fileStream);
```

它知道如何读取对象，但是要靠链接的stream提供文件存取

3 读取对象

```
Object one = os.readObject();
Object two = os.readObject();
Object three = os.readObject();
```

每次调用readObject()都会从stream中读出下一个对象，读取顺序与写入顺序相同，次数超过会抛出异常

4 转换对象类型

```
GameCharacter elf = (GameCharacter) one;
GameCharacter troll = (GameCharacter) two;
GameCharacter magician = (GameCharacter) three;
```

返回值是Object类型，因此必须要转换类型

5 关闭 ObjectInputStream

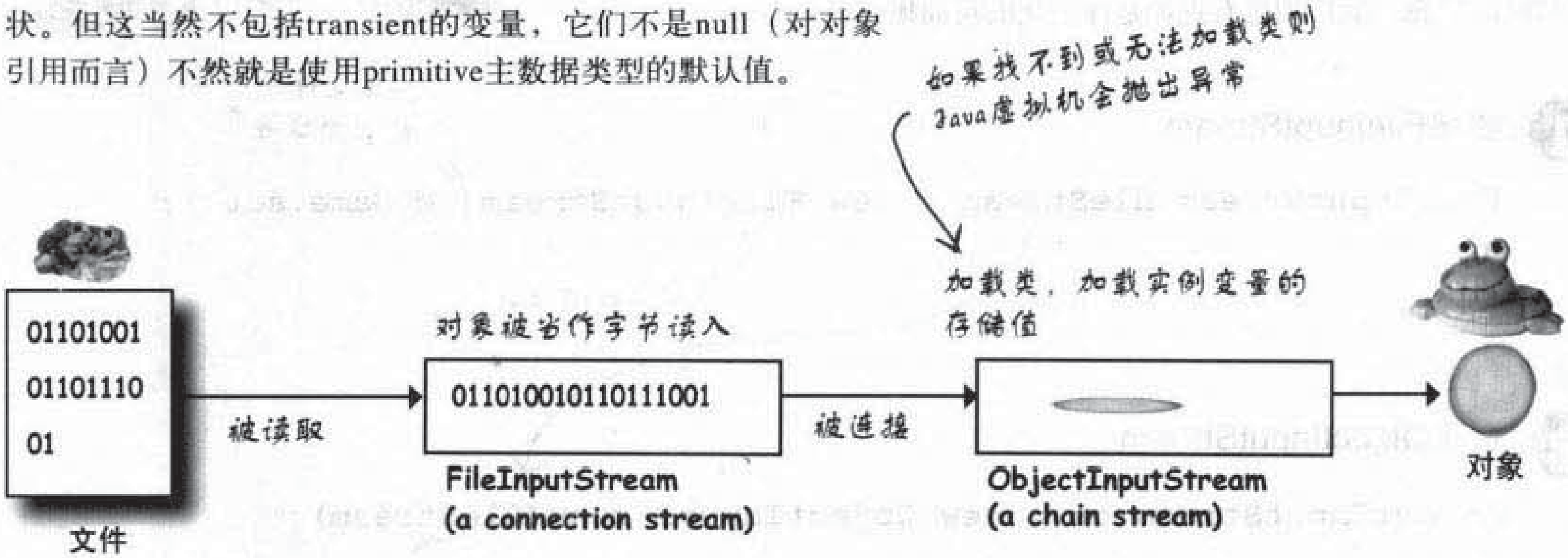
```
os.close();
```

FileInputStream会自动跟着关掉

PDG

解序列化的时候发生了什么事？

当对象被解序列化时，Java虚拟机会通过尝试在堆上创建新的对象，让它维持与被序列化时有相同的状态来恢复对象的原状。但这当然不包括transient的变量，它们不是null（对对象引用而言）不然就是使用primitive主数据类型的默认值。



1. 对象从stream中读出来。

2. Java虚拟机通过存储的信息判断出对象的class类型。

3. Java虚拟机尝试寻找和加载对象的类。如果Java虚拟机找不到或无法加载该类，则Java虚拟机会抛出例外。

4. 新的对象会被配置在堆上，但构造函数不会执行！很明显的，这样会把对象的状态抹去又变成全新的，而这不是我们想要的结果。我们需要的是对象回到存储时的状态。

5 如果对象在继承树上有个不可序列化的祖先类，则该不可序列化类以及在它之上的 类的构造函数（就算是可序列化也一样）就会执行。一旦构造函数连锁启动之后将无法停止。也就是说，从第一个不可序列化的父类开始，全部都会重新初始状态。

6 对象的实例变量会被还原成序列化时点的状态值。transient变量会被赋值null的对象引用或primitive主数据类型的默认为0、false等值。

there are no Dumb Questions

问： 为什么类不会存储成对象的一部分？这样就不会出现找不到类的问题了？

答： 当然也可以设计成这个样子，但这样是非常的浪费且会有很多额外的工作。虽然把对象序列化写在本机的硬盘上面不是什么很困难的工作，但序列化也有将对象送到网络联机上的用途。如果每个序列化对象都带有类，带宽的消耗可能就是个大问题。

对于通过网络传送序列化对象来说，事实上是有一种机制可以让类使用URL来指定位置。该机制用在Java的Remote Method Invocation（RMI，远程程序调用机制），让你可以把序列化的对象当作参数的一部分来传送，若接收此调用的Java虚拟机没有这个类的话，它可以自动地使用URL来取回并加载该类（第17章会讨论RMI）。

问： 那静态变量呢？它们会被序列化吗？

答： 不会。要记得static代表“每个类一个”而不是“每个对象一个”。当对象被还原时，静态变量会维持类中原本的样子，而不是存储时的样子。

存储与恢复游戏人物

创建人物……

```
import java.io.*;

public class GameSaverTest {
  public static void main(String[] args) {
    GameCharacter one = new GameCharacter(50, "Elf", new String[] {"bow", "sword", "dust"});
    GameCharacter two = new GameCharacter(200, "Troll", new String[] {"bare hands", "big ax"});
    GameCharacter three = new GameCharacter(120, "Magician", new String[] {"spells", "invisibility"});

    // 假设此处有改变人物状态值的程序代码

    try {
      ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream("Game.ser"));
      os.writeObject(one);
      os.writeObject(two);
      os.writeObject(three);
      os.close();
    } catch(IOException ex) {
      ex.printStackTrace();
    }
    one = null;
    two = null;
    three = null;
```

设定成null，因此无法存取堆上的这些对象

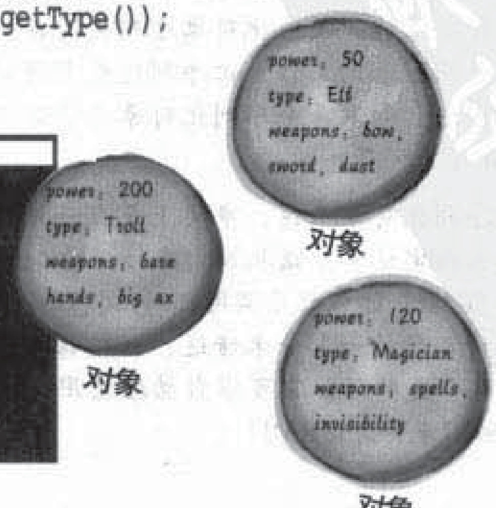
再从文件中把对象读回来

```
    try {
      ObjectInputStream is = new ObjectInputStream(new FileInputStream("Game.ser"));
      GameCharacter oneRestore = (GameCharacter) is.readObject();
      GameCharacter twoRestore = (GameCharacter) is.readObject();
      GameCharacter threeRestore = (GameCharacter) is.readObject();

      System.out.println("One's type: " + oneRestore.getType());
      System.out.println("Two's type: " + twoRestore.getType());
      System.out.println("Three's type: " + threeRestore.getType());
    } catch(Exception ex) {
      ex.printStackTrace();
    }
  }
}
```

看看是否成功

```
File Edit Window Help Resuscitate
% java GameSaver
Elf
Troll
Magician
```



GameCharacter类

```
import java.io.*;

public class GameCharacter implements Serializable {
   int power;
   String type;
   String[] weapons;

   public GameCharacter(int p, String t, String[] w) {
       power = p;
       type = t;
       weapons = w;
   }

   public int getPower() {
     return power;
   }

   public String getType() {
      return type;
   }

   public String getWeapons() {
       String weaponList = "";

       for (int i = 0; i < weapons.length; i++) {
          weaponList += weapons[i] + " ";
       }
       return weaponList;
   }
}
```

这是个测试序列化用的类，并没有实际的游戏功能

对象的序列化

要点

- 你可以通过序列化来存储对象的状态。
- 使用ObjectOutputStream来序列化对象（java.io）。
- Stream是连接串流或是链接用的串流。
- 连接串流用来表示源或目的地、文件、网络套接字连接。
- 链接用串流用来衔接连接串流。
- 用FileOutputStream链接ObjectOutputStream来将对象序列化到文件上。
- 调用ObjectOutputStream的writeObject(theObject)来将对象序列化，不需调用FileOutputStream的方法。
- 对象必须实现序列化这个接口才能被序列化。如果父类实现序列化，则子类也就自动地有实现，而不管是否有明确的声明。
- 当对象被序列化时，整个对象版图都会被序列化。这代表它的实例变量所引用的对象也会被序列化。
- 如果有不能序列化的对象，执行期间就会抛出异常。
- 除非该实例变量被标记为transient。否则，该变量在还原的时候会被赋予null或primitive主数据类型的默认值。
- 在解序列化时（deserialization），所有的类都必须能让Java虚拟机找到。
- 读取对象的顺序必须与写入的顺序相同。
- readObject()的返回类型是Object，因此解序列化回来的对象还需要转换成原来的类型。
- 静态变量不会被序列化，因为所有对象都是共享同一份静态变量值。

将字符串写入文本文件

通过序列化来存储对象是Java程序在来回执行间存储和恢复数据最简单的方式。但有时你还得把数据存储到单纯的文本文件中。假设你的Java程序必须把数据写到文本文件中以让其他可能是非Java的程序读取。例如你的servlet（在Web服务器上执行的Java程序）会读取用户在网页上输入的数据，并将它写入文本文件以让网站管理人能够用电子表格来分析数据。

写入文本数据（字符串）与写入对象是很类似的，你可以使用FileWrite来代替FileOutputStream（当然不会把它链接到ObjectOutputStream上）。

如果游戏人物数据写成常人可识别的文本文件大概就会像这样：

```
50,Elf,bow, sword,dust
200,Troll,bare hands,big ax
120,Magician,spells,invisibility
```

写序列化的对象：

```
objectOutputStream.writeObject(someObject);
```

写字符串：

```
fileWriter.write("My first String to save");
```

```
import java.io.*;        // 需要加载这个包

class WriteAFile {
    public static void main (String[] args) {

        try {
            FileWriter writer = new FileWriter("Foo.txt");   // 如果不存在就会被创建

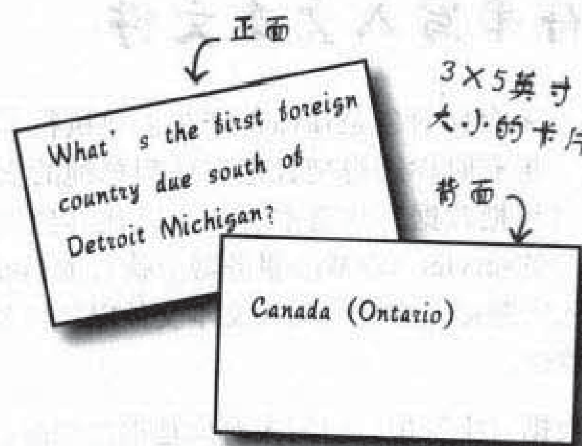
            writer.write("hello foo!");   // 以字符串作参数

            writer.close();   // 记得要关掉

        } catch(IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

输入/输出相关的操作都必须包在try/catch块中

文本文件范例：e-Flashcard



你知道老外在学校用的flashcard吗？那是一面有问题另外一面有答案的卡片。这种卡片对于理解式学习效果的帮助不大，但没有其他道具会比它还能加强机械化死背的记忆力。如果你想要来硬的，这倒还不错。拿来打发时间也还可以。

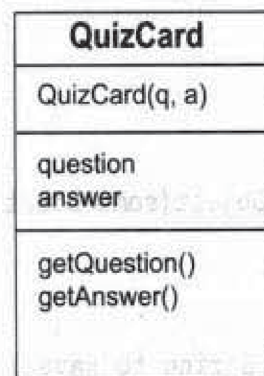
我们要用3个类来写个电子版flashcard：

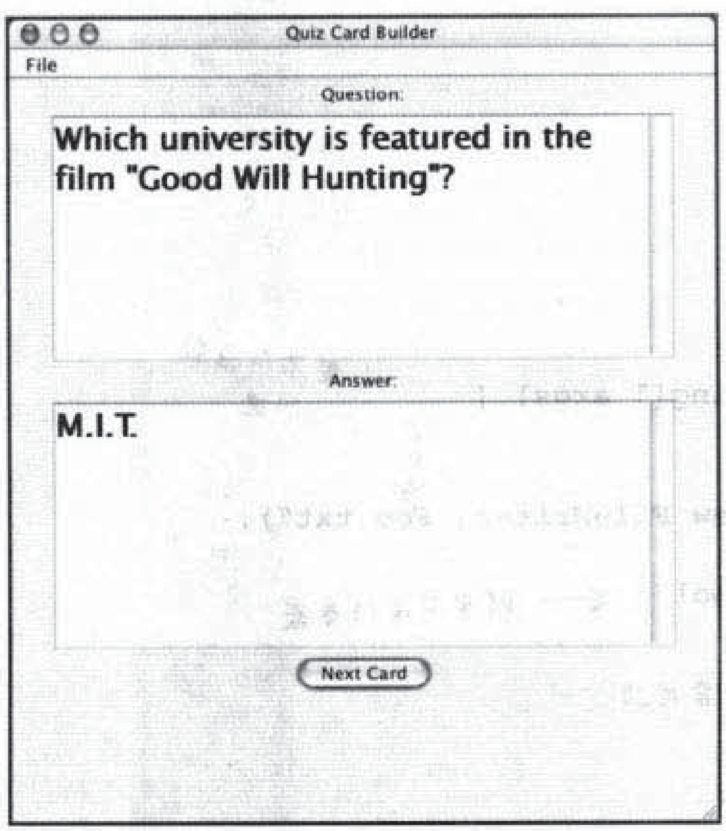
（1）QuizCardBuilder，设计并存储卡片的工具。

（2）QuizCardPlayer，加载并播放卡片的引擎。

（3）QuizCard，表示卡片数据的类。

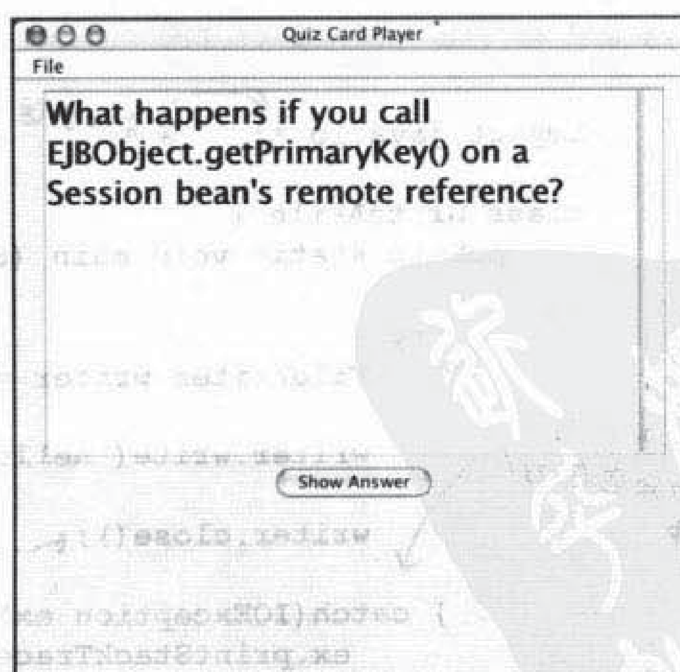
我们会查看builder和player的程序代码，并让你自己完成QuizCard，它会写成右边这样。





QuizCardBuilder

File菜单下有个Save选项能够让你存储一组数据到文本文件中。



QuizCardPlayer

File菜单下有个Load选项能够从文本文件加载一组卡片数据。

Quiz Card Builder（程序代码大纲）

```
public class QuizCardBuilder {

  public void go() {
    // 创建并显示gui
  }

  private class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // 向列表中增加当前卡片并清除文本域
    }
  }

  private class SaveMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // 生成对话框
      // 输入用户名并保存设置
    }
  }

  private class NewMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // 清除card列表和文本域
    }
  }

  private void saveFile(File file) {
    // 把列表输出到一个文本文件
  }
}
```

创建并显示GUI，包括事件监听者的设计和注册

内部类

按下"Next Card"时会被触发，代表用户完成卡片并继续下一张新卡片

内部类

会被菜单上的"Save"触发，代表用户想要存储目前这一组卡片

内部类

会被菜单上的"New"触发，这会打开新的一组卡片（还要清空文字块）

实际编写文件的程序

```
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class QuizCardBuilder {

    private JTextArea question;
    private JTextArea answer;
    private ArrayList<QuizCard> cardList;
    private JFrame frame;

    public static void main (String[] args) {
       QuizCardBuilder builder = new QuizCardBuilder();
       builder.go();
    }

    public void go() {
        // 创建gui

        frame = new JFrame("Quiz Card Builder");
        JPanel mainPanel = new JPanel();
        Font bigFont = new Font("sanserif", Font.BOLD, 24);
        question = new JTextArea(6,20);
        question.setLineWrap(true);
        question.setWrapStyleWord(true);
        question.setFont(bigFont);

        JScrollPane qScroller = new JScrollPane(question);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

        answer = new JTextArea(6,20);
        answer.setLineWrap(true);
        answer.setWrapStyleWord(true);
        answer.setFont(bigFont);

        JScrollPane aScroller = new JScrollPane(answer);
        aScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        aScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);


        JButton nextButton = new JButton("Next Card");

        cardList = new ArrayList<QuizCard>();

        JLabel qLabel = new JLabel("Question:");
        JLabel aLabel = new JLabel("Answer:");

        mainPanel.add(qLabel);
        mainPanel.add(qScroller);
        mainPanel.add(aLabel);
        mainPanel.add(aScroller);
        mainPanel.add(nextButton);
        nextButton.addActionListener(new NextCardListener());
        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenuItem newMenuItem = new JMenuItem("New");
```

所有的GUI程序都在这里，也许你要多注意一下有关Menu菜单的部分

```java
    JMenuItem saveMenuItem = new JMenuItem("Save");
    newMenuItem.addActionListener(new NewMenuListener());

    saveMenuItem.addActionListener(new SaveMenuListener());
    fileMenu.add(newMenuItem);
    fileMenu.add(saveMenuItem);
    menuBar.add(fileMenu);
    frame.setJMenuBar(menuBar);
    frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
    frame.setSize(500,600);
    frame.setVisible(true);
}

public class NextCardListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {

      QuizCard card = new QuizCard(question.getText(), answer.getText());
      cardList.add(card);
      clearCard();
   }
}

public class SaveMenuListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
      QuizCard card = new QuizCard(question.getText(), answer.getText());
      cardList.add(card);

      JFileChooser fileSave = new JFileChooser();
      fileSave.showSaveDialog(frame);
      saveFile(fileSave.getSelectedFile());
   }
}

public class NewMenuListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
      cardList.clear();
      clearCard();
   }
}

private void clearCard() {
   question.setText("");
   answer.setText("");
   question.requestFocus();
}

private void saveFile(File file) {
  try {
     BufferedWriter writer = new BufferedWriter(new FileWriter(file));

     for(QuizCard card:cardList) {
        writer.write(card.getQuestion() + "/");
        writer.write(card.getAnswer() + "\n");
     }
     writer.close();

  } catch(IOException ex) {
      System.out.println("couldn't write the cardList out");
      ex.printStackTrace();
  }
}
}
```

创建菜单，把new与save项目加到File下，然后指定frame使用这个菜单，菜单项目会触发ActionEvent

调出存盘对话框（dialog）等待用户决定，这都是靠JFileChooser完成的

实际编写文件的方法由SaveMenuListener的事件处理程序所调用，稍后会介绍File这个类

将BufferedWriter链接到FileWriter，稍后也有说明

将ArrayList中的卡片逐个写到文件中，一行一张卡片，问题和答案由"/"分开

java.io.File class

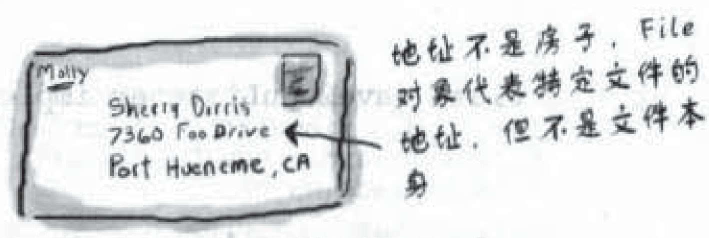
File这个类代表磁盘上的文件，但并不是文件中的内容。啥？你可以把File对象想象成文件的路径，而不是文件本身。例如File并没有读写文件的方法。关于File有个很有用的功能就是它提供一种比使用字符串文件名来表示文件更安全的方式。举例来说，在构造函数中取用字符串文件名的类也可以用File对象来代替该参数，以便检查路径是否合法等，然后再把对象传给FileWriter或FileInputStream。

File对象代表磁盘上的文件或目录的路径名称，如：

/Users/Kathy/Data/GameFile.txt

但它并不能读取或代表文件中的数据。

你可以对File对象做的事情：



① 创建出代表现存盘文件的File对象。

```
File f = new File("MyCode.txt");
```

② 建立新的目录。

```
File dir = new File("Chapter7");
dir.mkdir();
```

③ 列出目录下的内容。

```
if (dir.isDirectory()) {
      String[] dirContents = dir.list();
      for (int i = 0; i < dirContents.length; i++) {
          System.out.println(dirContents[i]);
      }
}
```

④ 取得文件或目录的绝对路径。

```
System.out.println(dir.getAbsolutePath());
```

⑤ 删除文件或目录（成功会返回true）。

```
boolean isDeleted = f.delete();
```

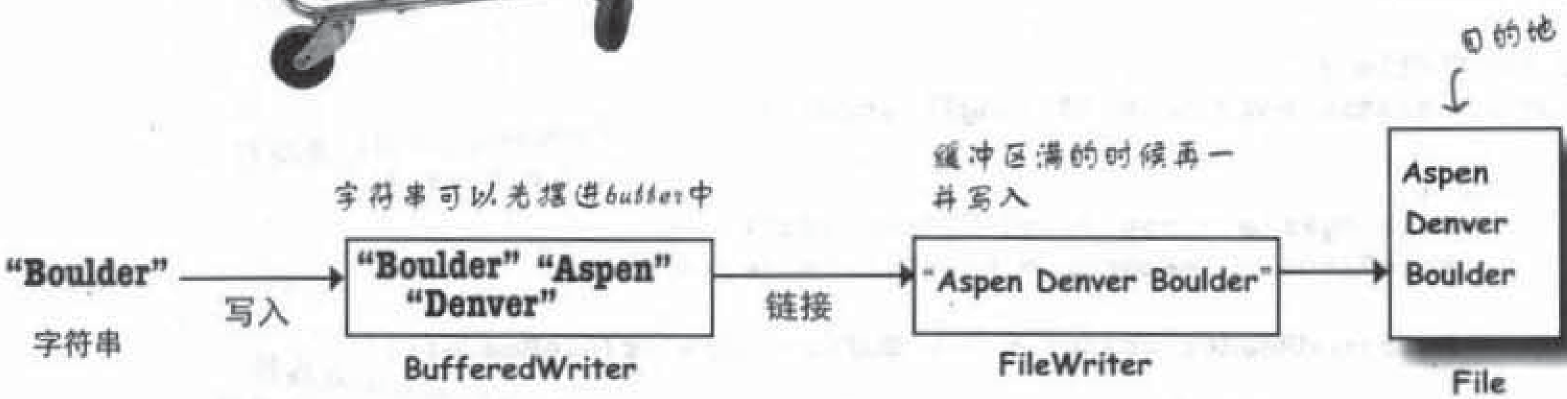


缓冲区的奥妙之处

没有缓冲区，就好像逛超市没有推车一样。你只能一次拿一项东西结账。



缓冲区能让你暂时摆一堆东西一直到满为止。用了缓冲区就可以省下好几趟的来回




```
BufferedWriter writer = new BufferedWriter(new FileWriter(aFile));
```

注意此处不需要持有对FileWriter对象的引用，我们只在乎BufferedWriter

缓冲区的奥妙之处在于使用缓冲区比没有使用缓冲区的效率更好。你也可以直接使用FileWriter，调用它的write()来写文件，但它每次都会直接写下去。你应该不会喜欢这种方式额外的成本，因为每趟磁盘操作都比内存操作要花费更多时间。通过BufferedWriter和FileWriter的链接，BufferedWriter可以暂存一堆数据，然后到满的时候再实际写入磁盘，这样就可以减少对磁盘操作的次数。

如果你想要强制缓冲区立即写入，只要调用writer.flush()这个方法就可以要求缓冲区马上把内容写下去。

读取文本文件

从文本文件读数据是很简单的，但是这次我们会使用File对象来表示文件，以FileReader来执行实际的读取，并用BufferedReader来让读取更有效率。

读取是以while循环来逐行进行，一直到readLine()的结果为null为止。这是最常见的读取数据方式（几乎非序列化对象都是这样的）：以while循环（实际上应该称为while循环测试）来读取，读到没有东西可以读的时候停止（通过读取结果为null来判断）。

带有两行的文本文件

```
What's 2 + 2?/4
What's 20+22/42
```

MyText.txt

```
import java.io.*;

class ReadAFile {
    public static void main (String[] args) {

        try {
            File myFile = new File("MyText.txt");
            FileReader fileReader = new FileReader(myFile);

            BufferedReader reader = new BufferedReader(fileReader);

            String line = null;

            while ((line = reader.readLine()) != null) {
                System.out.println(line);
            }
            reader.close();

        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

别忘了这个（指 import java.io.*;）

FileReader是字符的连接到文本文件的串流

将FileReader链接到BufferedReader以获取更高的效率。它只会在缓冲区读空的时候才会回头去磁盘读取

用String变量来承接所读取的结果

读一行就列出一行，直到没有东西可以读为止

Quiz Card Player（程序代码大纲）

```
public class QuizCardPlayer {

  public void go() {
    // 创建并显示gui
  }

  class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // 如果是个问题，显示答案，否则显示下一个问题
      // 改一个标识表明我们已经浏览了问题或答案
    }
  }

  class OpenMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
      // 生成一个文件对话框
      // 让用户把一个卡片设置打开
    }
  }

  private void loadFile(File file) {
     // 创建卡片的ArrayList，并从文本文件中读取它们
    // 调用OpenMenuListener事件处理器，每次从文件中读取一行
    // 告诉makeCard()方法创建一个新卡片
   // (one line in the file holds both the question and answer, separated by a "/")
  }


  private void makeCard(String lineToParse) {
    // 调用LoadFile方法，从文本文件中读取一行
    // 创建一个新的QuizCard，通过调用CardList把它加入ArrayList中
  }
}
```

Quiz Card Player代码

```
import java.util.*;
import java.awt.event.*;
import javax.swing.*;
import java.awt.*;
import java.io.*;

public class QuizCardPlayer {

    private JTextArea display;
    private JTextArea answer;
    private ArrayList<QuizCard> cardList;
    private QuizCard currentCard;
    private int currentCardIndex;
    private JFrame frame;
    private JButton nextButton;
    private boolean isShowAnswer;


    public static void main (String[] args) {
       QuizCardPlayer reader = new QuizCardPlayer();
       reader.go();
    }

    public void go() {

        // 创建gui

        frame = new JFrame("Quiz Card Player");
        JPanel mainPanel = new JPanel();
        Font bigFont = new Font("sanserif", Font.BOLD, 24);

        display = new JTextArea(10,20);
        display.setFont(bigFont);

        display.setLineWrap(true);
        display.setEditable(false);

        JScrollPane qScroller = new JScrollPane(display);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        nextButton = new JButton("Show Question");
        mainPanel.add(qScroller);
        mainPanel.add(nextButton);
        nextButton.addActionListener(new NextCardListener());

        JMenuBar menuBar = new JMenuBar();
        JMenu fileMenu = new JMenu("File");
        JMenuItem loadMenuItem = new JMenuItem("Load card set");
        loadMenuItem.addActionListener(new OpenMenuListener());
        fileMenu.add(loadMenuItem);
        menuBar.add(fileMenu);
        frame.setJMenuBar(menuBar);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(640,500);
        frame.setVisible(true);

    } // 关闭go
```

没什么，只是一般的GUI程序代码罢了

```
public class NextCardListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        if (isShowAnswer) {
            // 显示答案
            display.setText(currentCard.getAnswer());
            nextButton.setText("Next Card");
            isShowAnswer = false;
        } else {
            // 显示问题
            if (currentCardIndex < cardList.size()) {

                showNextCard();

            } else {
                // 没有更多的卡片了
                display.setText("That was last card");
                nextButton.setEnabled(false);
            }
        }
    }
}
```

检查isShowAnswer来判断现在看的是问题还是答案，并根据回答来执行适当的工作

```
public class OpenMenuListener implements ActionListener {
    public void actionPerformed(ActionEvent ev) {
        JFileChooser fileOpen = new JFileChooser();
        fileOpen.showOpenDialog(frame);
        loadFile(fileOpen.getSelectedFile());
    }
}
```

打开文件的对话框让用户选择文件

```
private void loadFile(File file) {

    cardList = new ArrayList<QuizCard>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(file));
        String line = null;
        while ((line = reader.readLine()) != null) {
            makeCard(line);
        }
        reader.close();

    } catch(Exception ex) {
        System.out.println("couldn't read the card file");
        ex.printStackTrace();
    }
    // 显示第一个卡片
}
```

读取一行数据，传给makeCard()来把字符串解析成卡片加到ArrayList中

```
private void makeCard(String lineToParse) {
    String[] result = lineToParse.split("/");
    QuizCard card = new QuizCard(result[0], result[1]);
    cardList.add(card);
    System.out.println("made a card");
}

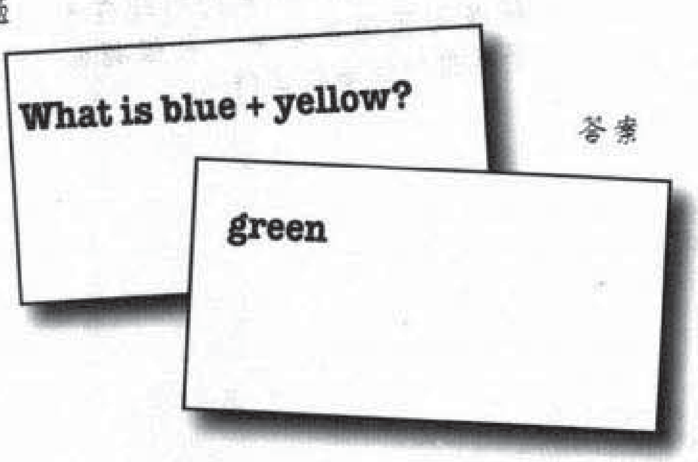
private void showNextCard() {
    currentCard = cardList.get(currentCardIndex);
    currentCardIndex++;
    display.setText(currentCard.getQuestion());
    nextButton.setText("Show Answer");
    isShowAnswer = true;
}
} // 关闭类
```

每一行都是一张卡片，但需要分解成问题和答案，这里使用到下一页会说明的split()

用String的split()解析

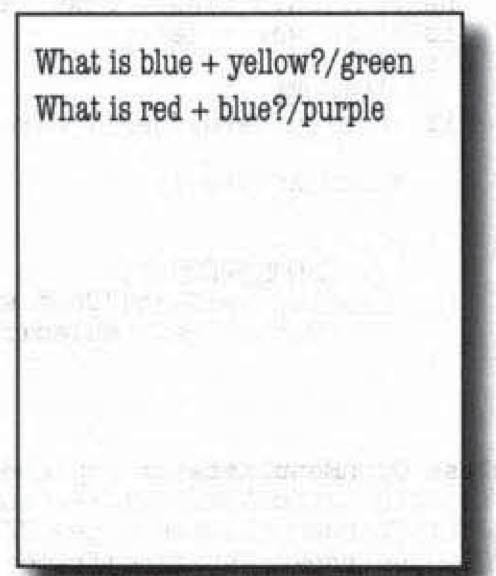
假设你的flashcard像下面这样：

问题



答案

文件中的问答像是下面这样：



要如何分开问题和答案？

当你读取文件时，问题和答案是合并在同一行，以“/”字符来分开的。

String的split()可以把字符串拆开。

split()可以将字符串拆开成String的数组。



单元一　　分隔字符　　单元二

从文件读出来的时候会像这样

```
String toTest = "What is blue + yellow?/green";
String[] result = toTest.split("/");
for (String token:result) {
   System.out.println(token);
}
```

split()会用参数所指定的字符来把这个String拆开成两个部分（此方法实际上能够做到比这个例子还要复杂的解析）

把数组中的每个元素逐一地列出来

there are no Dumb Questions

问： **我查看API的时候发现java.io这个包中有500万种类，你到底是怎么知道要用哪一种？**

答： 输入/输出的API使用一种模块化的“链接”概念来让你可以把连接串流与链接串流以各种可能应用到的排列组合连起来。

这个串流链并不是只能衔接两种层次，你可以连接多种链接串流来达成所需的处置。

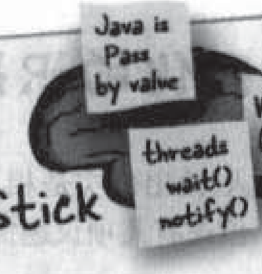
通常你只会用到少数几个类而已。如果想要读写文本文件，或许BufferedReader与BufferedWriter（链接到FileReader与FileWriter上）就够用了。

换言之，本书有涵盖到九成以上你所会用到的Java 输入/输出。

问： **那1.4版新增的输入/输出 nio这个类呢？**

答： java.nio这个类带来重大的效能提升并可以充分利用执行程序的机器上的原始容量。nio的一项关键能力是你可以直接控制buffer。另一项能力是non-blocking的输入/输出，它能让你的输入/输出程序代码在没有东西可读取或写入时不必等在那里。某些现有的类（包括FileInpuStream和FileOutputStream）会利用到其中一部分的功能。nio使用起来更为复杂，除非你真的很需要新功能，不然的话，使用我们这里所讨论的功能方法会简单得多。此外，如果没有很小心的设计，nio可能会引发效能损失。非nio的输入/输出适合九成以上的应用，特别在你还是新手的时候更是如此。

但你还是可以使用FileInputStream并通过getCahnnel()方法来存取channel来开始使用nio。



Make it Stick

玫瑰是红的，废话是很多的。

Reader和Writer只能用在文字上。

译者有话要说：原作者的观念也不是全部正确的，多写、多实验、多看些好书才能帮助你建立正确的观念。

要点

- 用FileWriter这个连接串流来写入文本文件。
- 将FileWriter链接到BufferedWriter可以提升效率。
- File对象代表文件的路径而不是文件本身。
- 你可以用File对象来创建、浏览和删除目录。
- 用到String文件名的串流大部分都可以用File对象来代替String。
- 用FileReader来读取文本文件。
- 将FileReader链接到BufferedReader可以提升效率。
- 通常我们会使用特殊的字符来分隔文本数据中的不同元素。
- 使用split()方法可以把String拆开，其中的分隔字符不会被当作数据来看待。

Version ID：序列化的识别

现在你已经知道Java的输入/输出确实是相当简单的，特别是很常见的连接/链接组合更是如此。但还有几项议题需要关注。

版本控制很重要！

如果你将对象序列化，则必须要有该类才能还原和使用该对象。OK，这是废话。但若你同时又修改了类会发生什么事？假设你尝试要把Dog对象带回来，而某个非transient的变量却已经从double被改成String。这样会很严重地违反Java的类型安全性。其实不只是修改会伤害兼容性，想想下列的情况：

会损害解序列化的修改：

删除实例变量。

改变实例变量的类型。

将非瞬时的实例变量改为瞬时的。

改变类的继承层次。

将类从可序列化改成不可序列化。

将实例变量改成静态的。

通常不会有事的修改：

加入新的实例变量（还原时会使用默认值）。

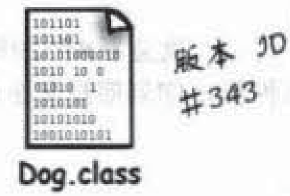
在继承层次中加入新的类。

从继承层次中删除类。

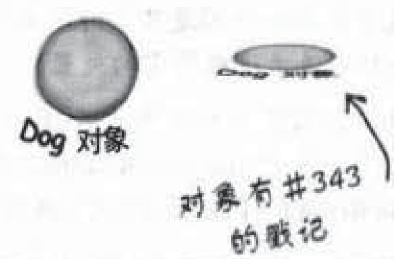
不会影响解序列化程序设定变量值的存取层次修改。

将实例变量从瞬时改成非瞬时（会使用默认值）。

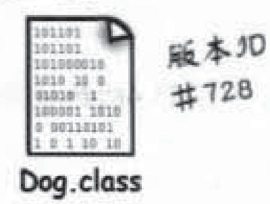
① 编写 Dog。



② 将Dog对象序列化。



③ 修改Dog。



④ 以修改过的类将Dog对象解序列化。



⑤ 还原失败！

俗话说得好：老狗变不出新把戏！

使用serialVersionUID

每当对象被序列化的同时，该对象（以及所有在其版图上的对象）都会被“盖”上一个类的版本识别ID。这个ID被称为serialVersionUID，它是根据类的结构信息计算出来的。在对象被解序列化时，如果在对象被序列化之后类有了不同的serialVersionUID，则还原操作会失败！但你还可以有控制权。

如果你认为类有可能会演化，就把版本识别ID放在类中。

当Java尝试要还原对象时，它会比对对象与Java虚拟机上的类的serialVersionUID。例如，如果Dog实例是以23这个ID来序列化的（实际的ID长得多），当Java虚拟机要还原Dog对象时，它会先比对Dog对象和Dog类的serialVersionUID。如果版本不相符，Java虚拟机就会在还原过程中抛出异常。

因此，解决方案就是把serialVersionUID放在class中，让类在演化的过程中还维持相同的ID。

这只会在你有很小心地维护类的变动时才办得到！也就是说你得要对带回旧对象的任何问题负起全责。

若想知道某个类的serialVersionUID，则可以使用Java Development Kit里面所带的serialver工具来查询。

```
File Edit Window Help serialKiller
% serialver Dog
Dog: static final long
serialVersionUID = -
5849794470654667210L;
```

当你的类可能会在产生序列化对象之后继续演进时……

① 使用serialver工具来取得版本ID。

```
File Edit Window Help serialKiller
% serialver Dog
Dog: static final long
serialVersionUID = -
5849794470654667210L;
```

② 把输出拷贝到类上。

```
public class Dog {

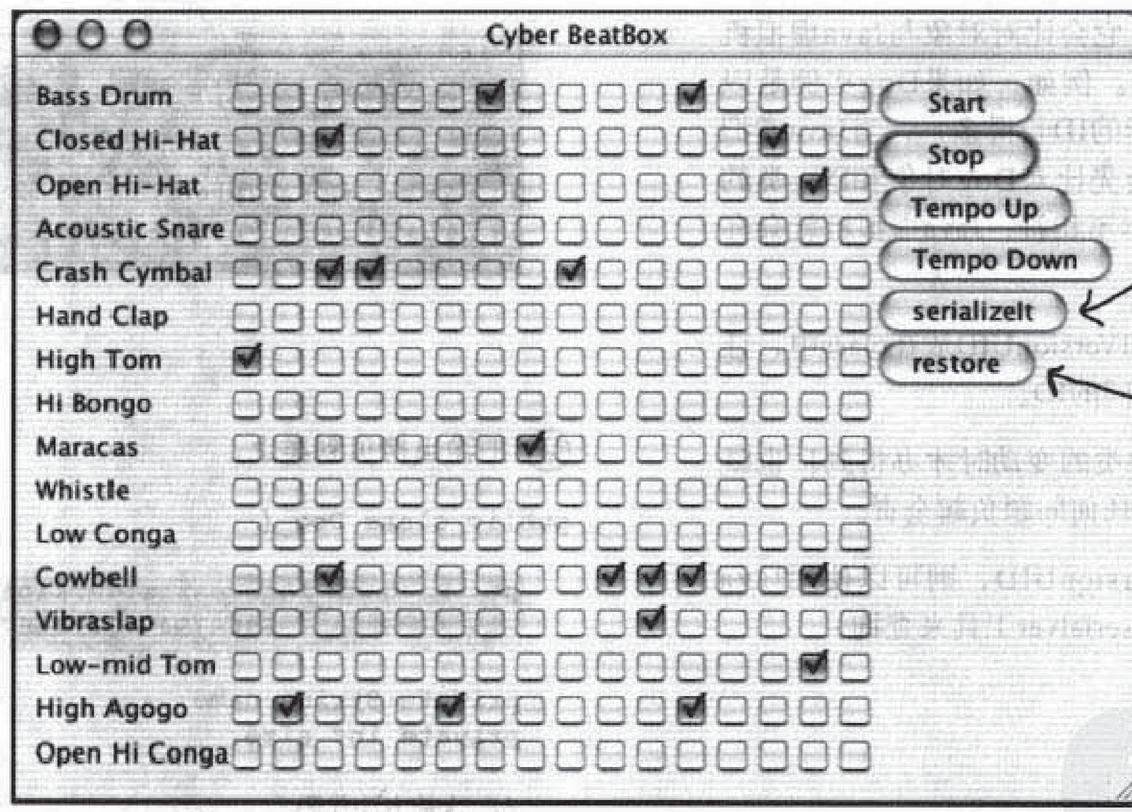
    static final long serialVersionUID =
                    -6849794470754667710L;

    private String name;
    private int size;

    // 这是方法代码
}
```

③ 在修改类的时候要确定修改程序的后果！例如，新的 Dog 要能够处理旧的 Dog 解序列化之后新加入变量的默认值。

程序料理



按下这个按钮会存储目前的节奏

它会加载节奏文件并重新设定复选框组

让BeatBox能够存储并加载节奏样式

存储BeatBox节奏

要记住，在BeatBox中的鼓声样式只不过是一组复选框而已。在播放sequence的同时，程序代码会逐个检查复选框以指出哪个鼓声应该在16拍之中放出。因此存储节奏对我们而言就是存储复选框的状态。

我们可以用简单的boolean数组来存储256个复选框的状态。只要其中的元素都是可被序列化的，数组对象就可以被序列化。因此存储数组是没有问题的。

要载回节奏样式时，只要读取单一的boolean数组对象（将它还原）并存回复选框就可以。你已经看过大部分的程序代码，因此这一章只会展示存储与还原的部分。

这段程序料理让我们可以为进入下一章作准备，下一章将不是写入文件，而是传送到网络上的服务器上，同时读取来源也不是文件，而是其他用户通过服务器来传送的。

这是个BeatBox程序代码中的内部类

```
public class MySendListener implements ActionListener {

    public void actionPerformed(ActionEvent a) {   ← 这会在用户按下按钮触发ActionEvent时执行

        boolean[] checkboxState = new boolean[256];   ← 此数组用来保存复选框的状态

        for (int i = 0; i < 256; i++) {

            JCheckBox check = (JCheckBox) checkboxList.get(i);   逐个取得状态并加到数组中
            if (check.isSelected()) {
               checkboxState[i] = true;
            }
        }

        try {
           FileOutputStream fileStream = new FileOutputStream(new File("Checkbox.ser"));
           ObjectOutputStream os = new ObjectOutputStream(fileStream);
           os.writeObject(checkboxState);
        } catch(Exception ex) {                          将boolean数组序列化
            ex.printStackTrace();
        }

      } // 关闭方法
    } // 关闭内部类
```

还原 BeatBox 节奏

这几乎就是存储的反向操作，读取boolean的数组然后使用它来还原GUI上复选框的状态。这都会发生在用户按下restore的按钮之后。

这是复选框程序代码中另外一个内部类

```
public class MyReadInListener implements ActionListener {

    public void actionPerformed(ActionEvent a) {
       boolean[] checkboxState = null;
       try {
           FileInputStream fileIn = new FileInputStream(new File("Checkbox.ser"));
           ObjectInputStream is = new ObjectInputStream(fileIn);
           checkboxState = (boolean[]) is.readObject(); ← 读取文件中的对象并将读取回来的Object类型转换回boolean数组

       } catch(Exception ex) {ex.printStackTrace();}


       for (int i = 0; i < 256; i++) {
          JCheckBox check = (JCheckBox) checkboxList.get(i);
          if (checkboxState[i]) {              还原每个checkbox的状态
             check.setSelected(true);
          } else {
             check.setSelected(false);
          }
       }

       sequencer.stop();           停止目前播放的节奏并使用复选框状
       buildTrackAndStart();       态重新创建序列

    } // 关闭方法
} // 关闭内部类
```

Sharpen your pencil

这是个有很大限制的版本。当你按下serializeIt按钮时，它会自动地将节奏序列化到Checkbox.ser这个文件中盖掉旧的。

通过引入JFileChooser就能够加强存储与回复的功能，让你能够为文件命名，也能自由地选择所要的文件。

Sharpen your pencil

谁能被存储？

下列哪种对象你认为应该要是可序列化的？如果不是，又为何？没有意义吗？安全风险吗？只对目前执行中的Java虚拟机有用？猜猜看，先不要偷看API文件。

类型	可序列化吗	如果不行，是什么原因？
Object	Yes / No	______________
String	Yes / No	______________
File	Yes / No	______________
Date	Yes / No	______________
OutputStream	Yes / No	______________
JFrame	Yes / No	______________
Integer	Yes / No	______________
System	Yes / No	______________

找碴！?

圈选出右边可以通过编译的程序片段。




```
FileReader fileReader = new FileReader();
BufferedReader reader = new BufferedReader(fileReader);
```

```
FileOutputStream f = new FileOutputStream(new File("Foo.ser"));
ObjectOutputStream os = new ObjectOutputStream(f);
```

```
BufferedReader reader = new BufferedReader(new FileReader(file));
String line = null;
while ((line = reader.readLine()) != null) {
    makeCard(line);
}
```

```
ObjectInputStream is = new ObjectInputStream(new FileOutputStream("Game.ser"));
GameCharacter oneAgain = (GameCharacter) is.readObject();
```



这一章探索了美好的Java I/O世界。你的任务是判断下列关于I/O的陈述是否为正确的叙述。

是非题

(1) 序列化适用于存储数据给非 Java 程序使用的情境。

(2) 对象的状态只能用序列化来存储。

(3) ObjectOutputStream是个用来存储序列化对象的类。

(4) 链接串流可以单独使用或与链接串流并用。

(5) 调用writeObject()可能会存储好几个对象。

(6) 所有的类都是默认为可序列化的。

(7) transient这个修饰符让你将实例变量标记为可序列化的。

(8) 如果父类是不可序列化的，则子类就不能设定成可序列化。

(9) 当对象被序列化时，它的读取顺序必须是相反的。

(10) 当对象还原时，构造函数不会执行。

(11) 序列化与使用文本文件保存的操作都可能会抛出异常。

(12) BufferedWriter可以链接到FileWriter上。

(13) File对象代表文件而不是目录。

(14) 你无法强制缓冲区写出数据。

(15) 文件的读写串流机制可以用到缓冲区机制。

(16) String的split()会将分隔字符解析成结果的一部分。

(17) 对类的任何修改都会破坏之前的序列化对象。

排排看

下面是被打乱的Java程序片段。你是否能够将它们重新排列以成为可以编译与执行并产生如同下方的输出结果？不一定每个片段都会用到或只能用一次。

```
class DungeonGame implements Serializable {
```

```
try {
```

```
FileOutputStream fos = new
   FileOutputStream( dg.ser );
```

```
short getZ() {
   return z;
```

```
   e.printStackTrace();
```

```
oos.close();
```

```
ObjectInputStream ois = new
   ObjectInputStream(fis);
```

```
int getX() {
   return x;
```

```
System.out.println(d.getX()+d.getY()+d.getZ());
```

```
FileInputStream fis = new
   FileInputStream( dg.ser );
```

```
public int x = 3;
transient long y = 4;
private short z = 5;
```

```
long getY() {
   return y;
```

```
class DungeonTest {
```

```
ois.close();
```

```
import java.io.*;
```

```
fos.writeObject(d);
```

```
} catch (Exception e) {
```

```
d = (DungeonGame) ois.readObject();
```

```
File Edit Window Help Torture
% java DungeonTest
12
8
```

```
ObjectOutputStream oos = new
   ObjectOutputStream(fos);
```

```
oos.writeObject(d);
```

```
public static void main(String [] args) {
   DungeonGame d = new DungeonGame();
```

解答

（1）序列化适用于存储数据给非 Java 程序使用的情境。	False
（2）对象的状态只能用序列化来存储。	False
（3）ObjectOutputStream是个用来存储序列化对象的类。	True
（4）链接串流可以单独使用或与链接串流并用。	False
（5）调用writeObject()可能会存储好几个对象。	True
（6）所有的类都是默认为可序列化的。	False
（7）transient这个修饰符让你将实例变量标记为可序列化的。	False
（8）如果父类是不可序列化的，则子类就不能设定成可序列化。	False
（9）当对象被序列化时，它的读取顺序必须是相反的。	False
（10）当对象还原时，构造函数不会执行。	True
（11）序列化与使用文本文件保存的操作都可能会抛出异常。	True
（12）BufferedWriter可以链接到FileWriter上。	True
（13）File对象代表文件而不是目录。	False
（14）你无法强制缓冲区写出数据。	False
（15）文件的读写串流机制可以用到缓冲区机制。	True
（16）String的split()会将分隔字符解析成结果的一部分。	False
（17）对类的任何修改都会破坏之前的序列化对象。	False





```
import java.io.*;

class DungeonGame implements Serializable {
  public int x = 3;
  transient long y = 4;
  private short z = 5;
  int getX() {
    return x;
  }
  long getY() {
    return y;
  }
  short getZ() {
    return z;
  }
}

class DungeonTest {
  public static void main(String [] args) {
    DungeonGame d = new DungeonGame();
    System.out.println(d.getX() + d.getY() + d.getZ());
    try {
      FileOutputStream fos = new FileOutputStream("dg.ser");
      ObjectOutputStream oos = new ObjectOutputStream(fos);
      oos.writeObject(d);
      oos.close();
      FileInputStream fis = new FileInputStream("dg.ser");
      ObjectInputStream ois = new ObjectInputStream(fis);
      d = (DungeonGame) ois.readObject();
      ois.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
    System.out.println(d.getX() + d.getY() + d.getZ());
  }
}
```

```
File Edit Window Help Escape
% java DungeonTest
12
8
```

15 网络与线程

网络联机



连接到外面的世界。 你的Java程序可以向外扩展并触及其他计算机上的程序。这很容易，所有网络运作的低层细节都已经由java.net函数库处理掉了。Java的一项好处是传送与接收网络上的数据只不过是链接上使用不同链接串流的输入/输出而已。如果有 BufferedReader就可以读取。若数据来自文件或网络另一端，则BufferedReader不用花费很多精力去照顾。在这一章中，我们会使用socket来连接外面的世界。我们会创建客户端的socket、服务器端的socket，并且会让两端相互交谈。在完成这一章的进度之前，你会创建出功能完整、多线程的聊天程序客户端。咦？我刚刚说multithreaded吗？好吧，你还会学习到如何一边开车、一边吃盒饭、一边看报纸、一边聊天、一边修指甲……

实时BeatBox聊天程序

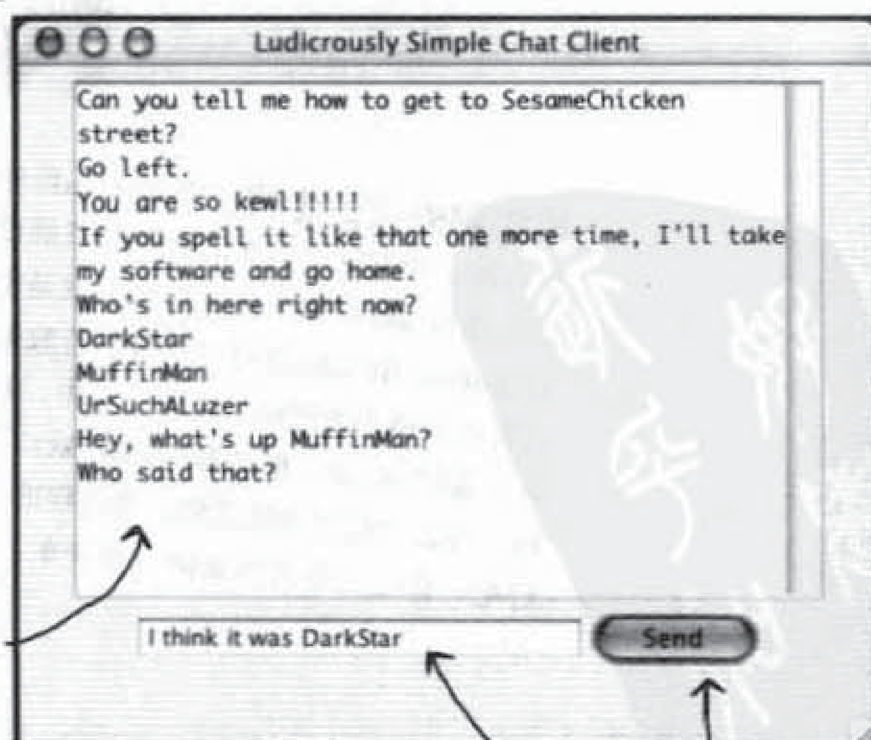


输入信息并按下sendIt按钮就会把信息和节奏样式送出

点选接收到的信息就会加载它的样式

你正在设计计算机游戏，同事们正在进行每个关卡的音效设计。使用有“聊天”功能版的BeatBox，团队成员就可以协同工作——你可以在讨论的同时直接送出节奏样式，让上线讨论的每个人都可以收到。因此你不只能读取他人的意见，还可以借着点选讨论信息区域来加载和播放节奏样式。

在这一章我们会学习到怎样作出这样的聊天程序。我们还会讨论到聊天服务器的制作。完整的BeatBox聊天程序会收录在“程序料理”一节，但前面的内容部分会来开发出“傻瓜版”客户端聊天程序与“限量发行流行时尚品味生活高级电动多功能后现代主义极简版”的聊天服务器来传送与接收文字信息。



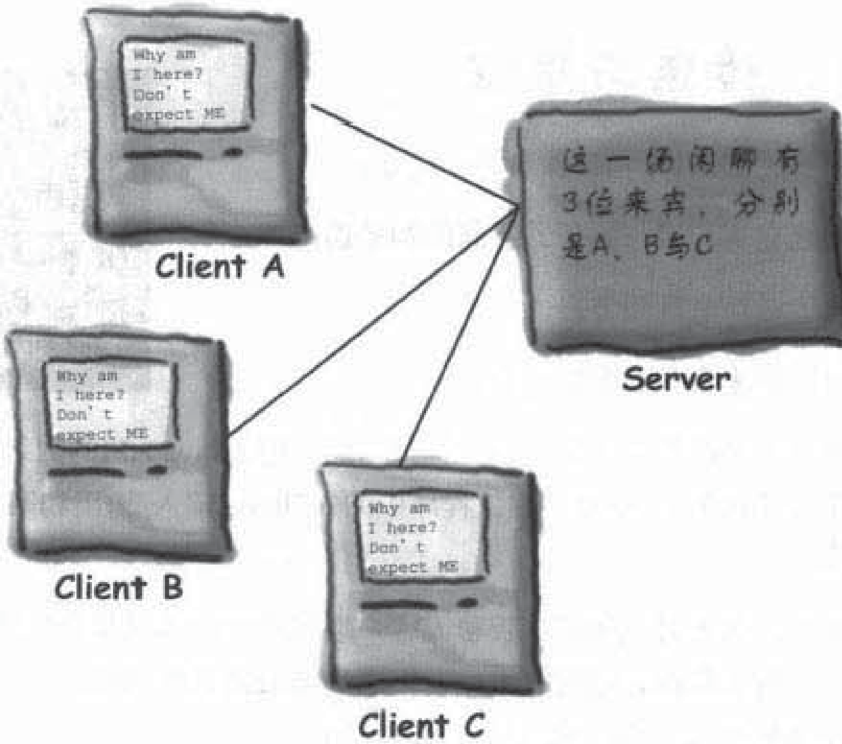
此处通常会进行充满智慧和善意的匿名具妙对话，每个人都会看到！

将你的信息送到服务器上

聊天程序概述

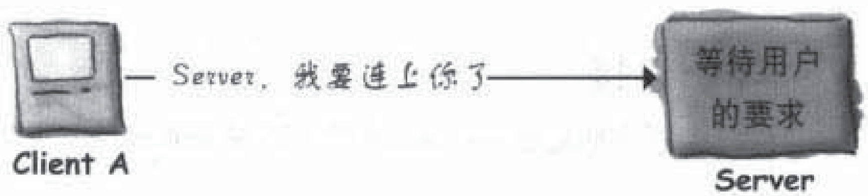
客户端必须要认识服务器。

服务器必须要认识所有的客户端。

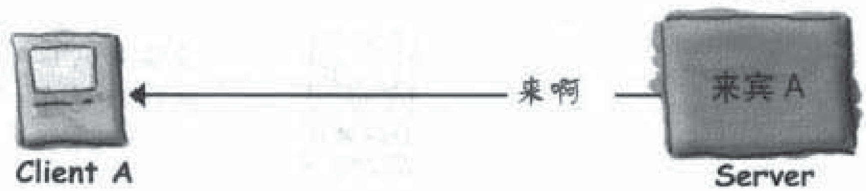


工作方式：

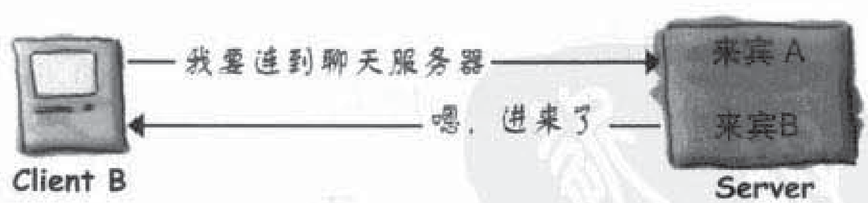
❶ 客户端连接到服务器。



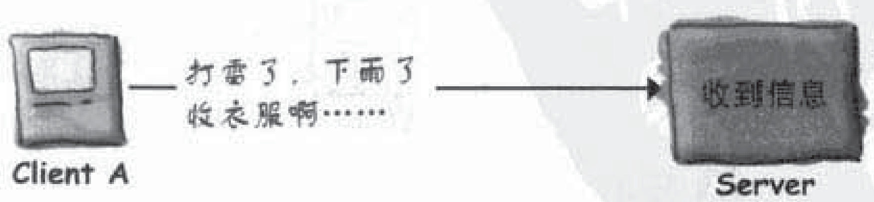
❷ 服务器建立连接并把客户端加到来宾清单中。



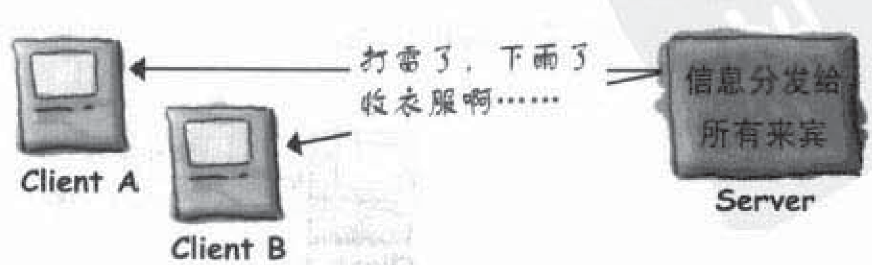
❸ 另外一个用户连接上来。



❹ 用户 A 送出信息到聊天服务器上。



❺ 服务器将信息送给所有的来宾。



连接、传送与接收

*译者个人意见：networking的学问绝对不是三言两语说得完，你也许可以很轻松地写出有网络功能的程序，但没有打好底子（像是深入研究TCP/IP等协议的原理）就很容易出问题（例如说效率不好）同时你也不会有除错抓问题的能力

要让客户端能够工作，有3件事必须先学：

（1）如何建立客户端与服务器之间的初始连接

（2）如何传送信息到服务器。

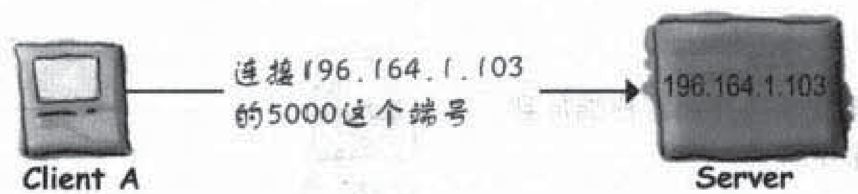
（3）如何接收来自服务器的信息。

这里面有非常多的低层工作细节。但很幸运，因为Java API的网络功能包（java.net）能让程序员轻松解决这些问题*。程序中的GUI码比输入/输出和网络码多了不少的原因就在此。

不只是这样，有个潜藏在简单版聊天客户端程序中的问题是我们还没遇过的：同时做两件事。建立联机是单次的工作。但之后聊天来宾得同时处理传送和接收信息，这得好好想个办法，我们稍后就会加以说明。

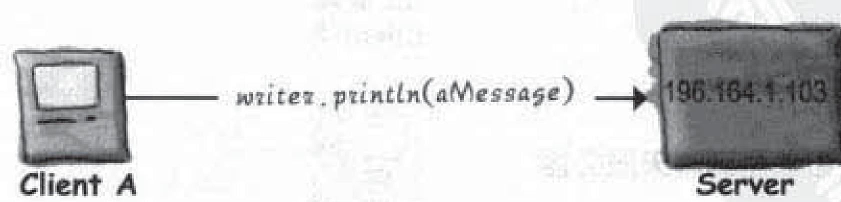
❶ 连接

用户通过建立socket连接来连接服务器。



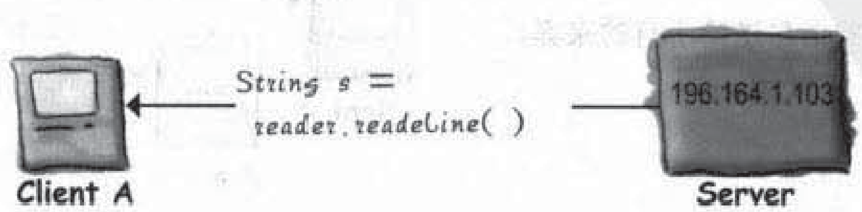
❷ 传送

用户送出信息给服务器。



❸ 接收

用户从服务器接收信息。



建立Socket连接

要连接到其他的机器上，我们会需要Socket的连接。Socket是个代表两台机器之间网络连接的对象（java.net.Socket）。什么是连接？两台机器之间的一种关系，让两个软件相互认识对方。最重要的是两者知道如何与对方通信，也就是说知道如何发送数据给对方。

还好我们不在乎低层的细节，因为这是在低层的“网络设备”中处理的。如果你不知道什么是网络设备也没关系，它只是一种让运行在Java虚拟机上的程序能够找到方法去通过实际的硬件（例如说网卡）在机器之间传送数据的机制。反正有人会负责这些低层的工作就对了。这些人是由操作系统的特定部分与 Java的网络API所组成的。你所必须要处理的是高层的部分——真的很高级，且又简单到很吓人。准备好面对这一切了吗？

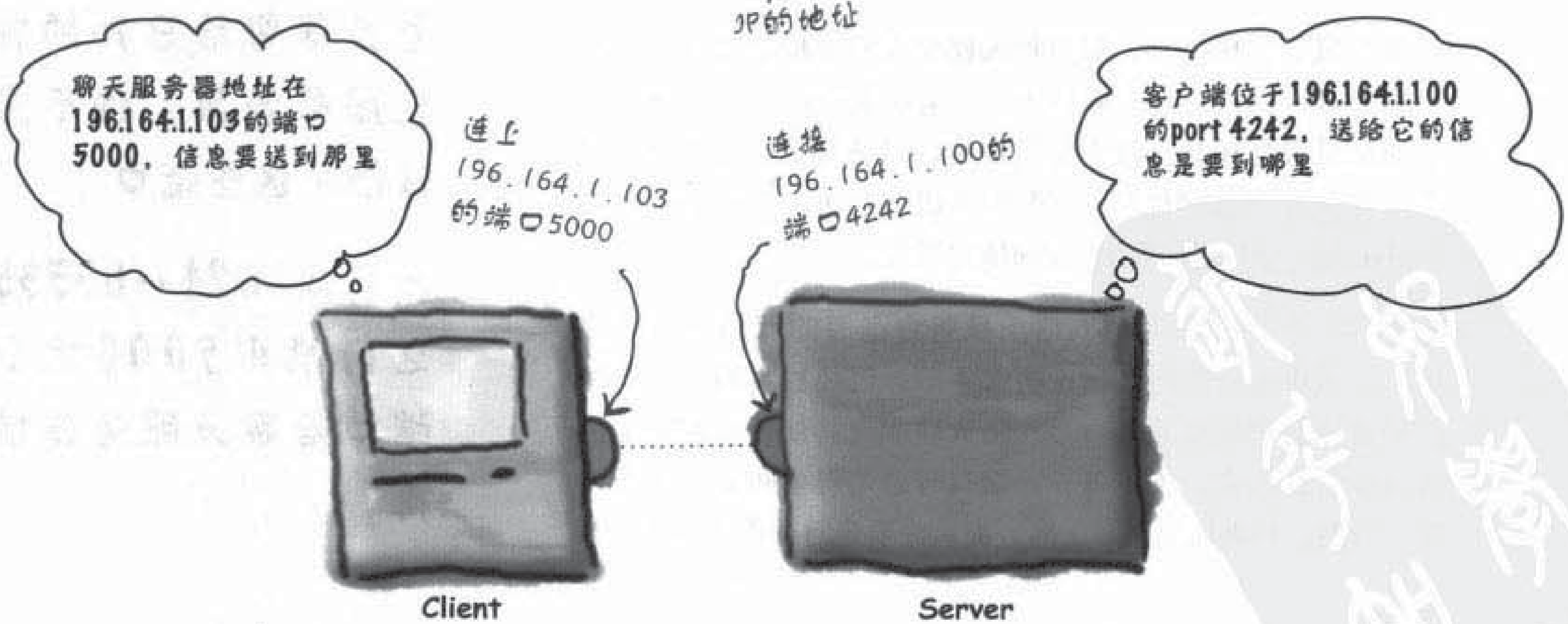
要创建Socket连接你得知道两项关于服务器的信息：它在哪里以及用哪个端口来收发数据。

也就是说IP地址与端口号。

TCP的端口号

```
Socket chatSocket = new Socket("196.164.1.103", 5000);
```

IP的地址



Socket连接的建立代表两台机器之间存有对方的信息，包括网络地址和TCP的端口号。

TCP端口只是一个16位宽、用来识别服务器上特定程序的数字

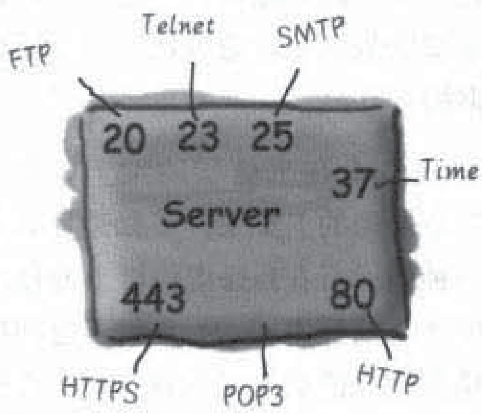
网页服务器（HTTP）的端口号是80，这是规定的标准。如果你有Telnet服务器的话，它的端口号会是23，POP3邮件服务器的是110，SMTP邮局交换服务器是25。把这些数字想成识别的代号（就像华安的终身代号是9527一样）。它们代表在服务器上执行软件的逻辑识别。注意，你在机器的背后找不到这些端口插孔。每个服务器上都有65536个端口（0～65535），很明显，哪来那么多实际的端口可以用。它只是个逻辑上用来表示应用程序的数字。

如果没有端口号，服务器就无法分辨客户端是要连到哪个应用程序的服务。每个应用程序可能都有独特的工作交谈方法，如果没有识别就发送的话会制造很大的混乱。就像是对邮件服务器发送HTTP请求。

在编写服务器程序时，你会加入程序代码来指定想要使用哪个端口号（稍后会有实际例子）。对聊天程序来说，我们选择的端口号是5000。没有特定理由，只是我们很喜欢这个数字，并且这个数字也介于1024～65535之间。为什么？因为0～1023都已经被保留给已知的特定服务。

如果你打算要在公司的网络上执行自己写的服务（服务程序），就得要先跟网管讨论有哪些端口已经被占用了。有时候网管也会把特定的端口号用防火墙或者其他特定的安全控管机制封锁起来。不管怎样，端口号总是得要挑选一个可用的。当然，如果你是在家上网，那就得要先问过你的小孩。

常见的TCP端口号。



一个地址可以有65536个不同的端口号可用。

从0～1023的TCP端口号是保留给已知的特定服务使用，你不应该使用这些端口*。

我们从1024～65535之间挑出5000这个端口给聊天服务器使用。

*也有可能你比网管大哥更厉害，此时爱用哪个端口当然是你说了算。

there are no Dumb Questions

问： 你怎么知道要跟哪个端口沟通？

答： 这要看程序是否为已知的服务。如果你尝试要连接众所周知的服务，像是HTTP、FTP、SMTP等，你可以在网络上用“已知tcp端口”来搜索引擎。或者用嘴问，别人也会叫你自己去查……

但如果程序不是这一类有共同标准的，你就得问谁开发的此服务，或者查询相关文件。这通常要问原来写程序或者是安装设定的人。例如你想要写个MSN Messenger的程序，可以上MSDN网站查相关资料。

问： 不同程序可以共享一个端口吗？

答： 不行，如果你想要使用（技术上叫做绑定）某个已经被占用的端口，就会收到BindException。绑定一个端口就代表程序要在特定的端口上面执行。

后面还有关于服务器的讨论会有更多相关的细节信息。

IP地址就好像是门牌号码



IP地址可以指定特定的地方，像是“展览路1号”

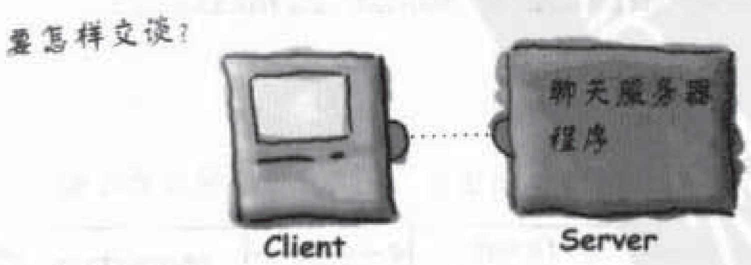
端口号就是该地址不同的窗口



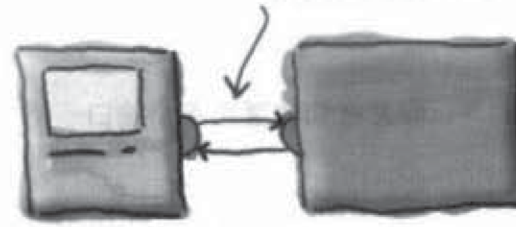
25号窗口处理邮件，80号窗口处理网页

Brain Barbell

现在你已经能够建立Socket连接。客户端与服务器都已经知道对方的IP地址和端口号。接下来呢？要如何通信？如何传送数据？



使用 BufferedReader 从 Socket 上读取数据



用串流来通过Socket连接来沟通。它是跟上一章所用相同的一般串流。Java的好处就在于大部分的输入/输出工作并不在乎链接串流的上游实际上是什么。也就是说你可以使用BufferedReader而不管是串流来自文件或Socket。

❶ 建立对服务器的Socket连接

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

用我们所说的5000号端口

127.0.0.1这个IP地址有特殊意义，就是“本机”，所以可以在自己这台计算机上同时测试客户端和服务器

❷ 建立连接到Socket上低层输入串流的InputStreamReader

```
InputStreamReader stream = new InputStreamReader(chatSocket.getInputStream());
```

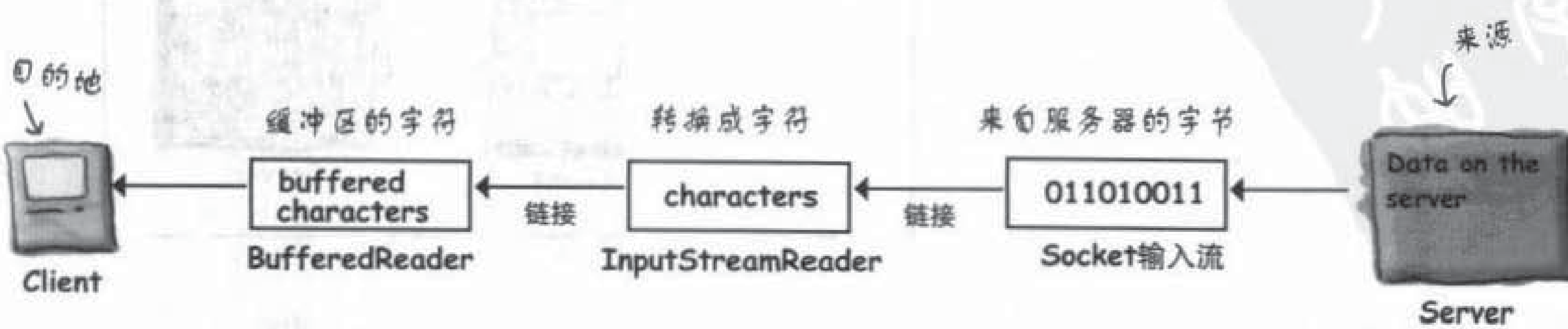
这是个低层和高层串流间的桥梁

从Socket取得输入串流

❸ 建立BufferedReader来读取

```
BufferedReader reader = new BufferedReader(stream);
String message = reader.readLine();
```

将BufferedReader链接到InputStreamReader



用PrintWriter写数据到Socket上

上一章没有使用到PrintWriter，而是用BufferedWriter。现在有了别的选择，但因为每次都是写入一个String，所以PrintWriter是最标准的做法。并且PrintWriter中有print()和println()方法就跟System.out里面的两个刚好一样！

1 对服务器建立Socket连接

这部分跟上一页相同

```
Socket chatSocket = new Socket("127.0.0.1", 5000);
```

2 建立链接到Socket的PrintWriter

```
PrintWriter writer = new PrintWriter(chatSocket.getOutputStream());
```

字符数据和字节间的转换桥梁，可以衔接String和Socket两端

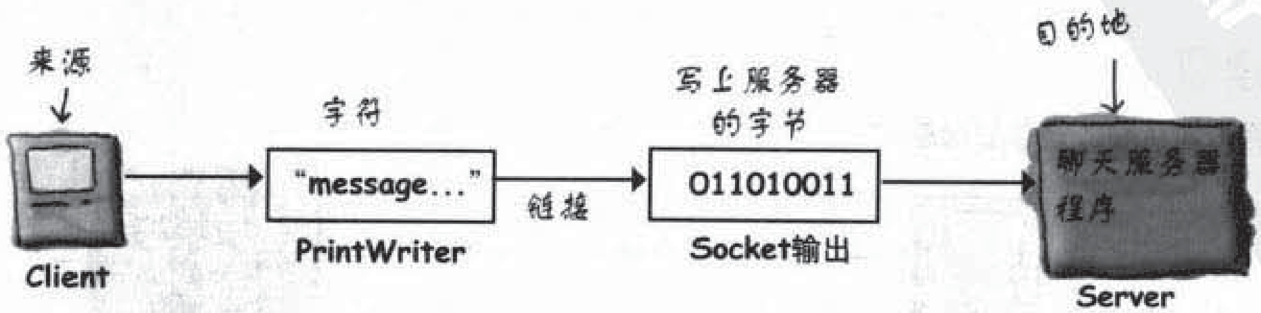
传给这个constructor的参数是来自于Socket

3 写入数据

```
writer.println("message to send");
writer.print("another message");
```

println()会在送出的数据后面加上换行

print()不会加上换行



每日嘉言

在我们开始创建聊天程序之前，先做一个小程序。“叶教授”是一位充满人生智慧哲理的大师，它能在你单调孤寂的程序设计生活中指引、启发你脆弱又容易受伤的心灵。

我们会创建一个称为“The Advice Guy”的程序，它能在启动的时候从服务器上提取叶教授的嘉言精华来滋润你干枯的胡渣。

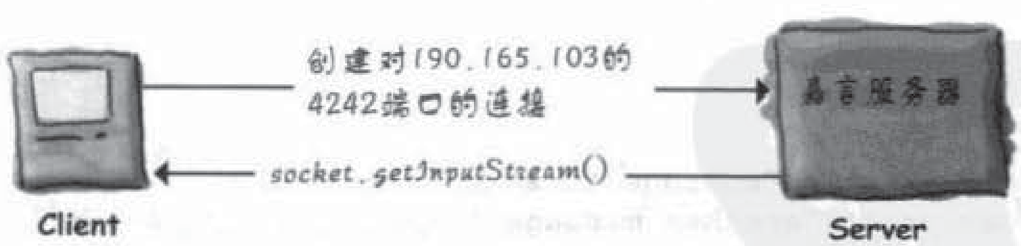
这么好的事情你还等什么？心动不如马上行动，如在通话中请稍后再拨。



叶教授

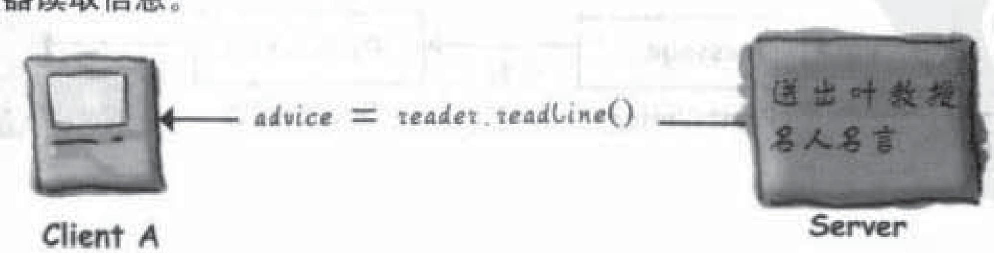
❶ 连接

客户端连上服务器并取得输入串流。



❷ 读取

客户端从服务器读取信息。



DailyAdviceClient客户端程序

这个程序会建立Socket，通过其他串流来制作BufferedReader，并从服务器应用程序（用4242端口服务的任何程序）上读取一行信息。

```
import java.io.*;
import java.net.*;        ← Socket是在java.net下面

public class DailyAdviceClient {

   public void go() {
       try {   ← 有可能出状况                ↙ 对端口4242作连接
          Socket s = new Socket("127.0.0.1", 4242);

          InputStreamReader streamReader = new InputStreamReader(s.getInputStream());
          BufferedReader reader = new BufferedReader(streamReader);  ← 链接数据串流



          String advice = reader.readLine();   ← 这个readLine()就跟读取文件
          System.out.println("Today you should: " + advice);   来源是一样的

          reader.close(); ← 会关闭所有的串流

      } catch(IOException ex) {
          ex.printStackTrace();
      }
   }

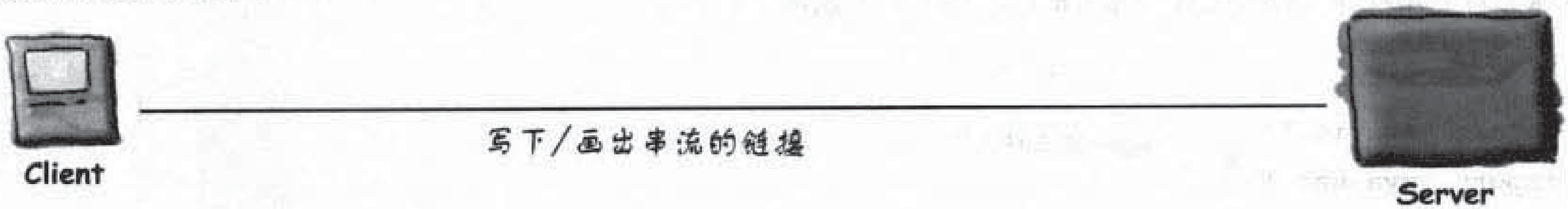
   public static void main(String[] args) {
       DailyAdviceClient client = new DailyAdviceClient();
       client.go();
   }
}
```

Sharpen your pencil

测试一下你对Socket读写所需的串流的记忆力。不要偷看上一页。

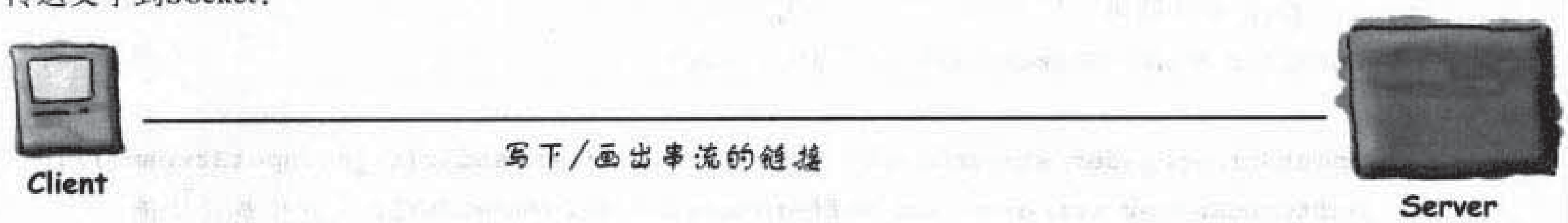
来源

从Socket读取文字：



目的地

传送文字到Socket：



Sharpen your pencil

问答题

（1）建立Socket连接时需要知道服务器的哪两项信息？

（2）HTTP与FTP等众所周知的服务会用到哪个端口？

（3）有效的TCP端口共有多少个？

编写简单的服务器程序

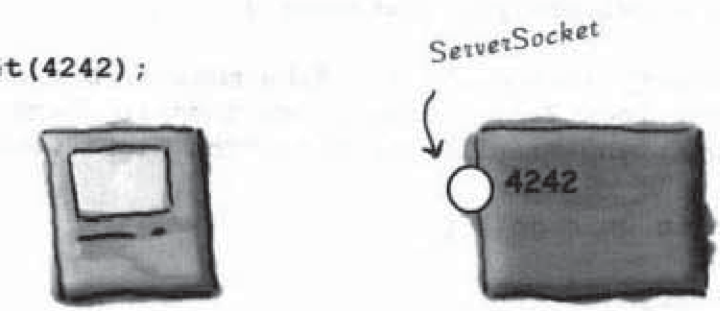
编写服务器应用程序要用到哪些东西呢？一对Socket。没错，两个称为一对。它们是一个会等待用户请求（当用户创建Socket 时）的ServerSocket以及与用户通信用的Socket。

工作方式

1 服务器应用程序对特定端口创建出ServerSocket。

```
ServerSocket serverSock = new ServerSocket(4242);
```

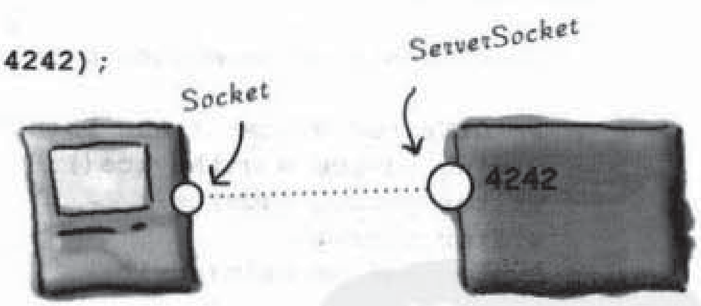
这会让服务器应用程序开始监听来自4242端口的客户端请求。



2 客户端对服务器应用程序建立Socket连接。

```
Socket sock = new Socket("190.165.1.103", 4242);
```

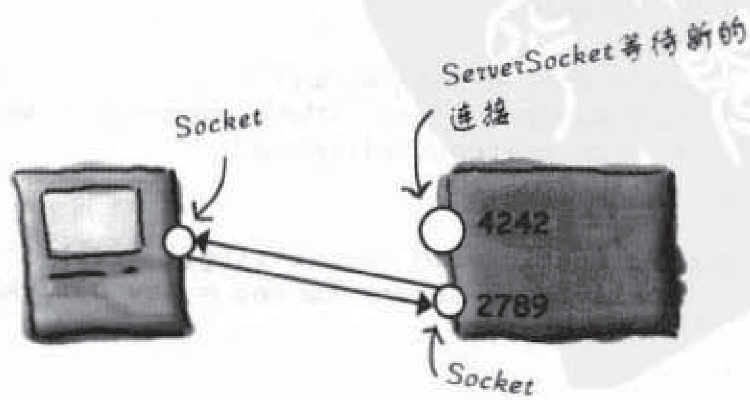
客户端得知道 IP 地址与端口号。



3 服务器创建出与客户端通信的新Socket。

```
Socket sock = serverSock.accept();
```

accept()方法会在等待用户的Socket连接时闲置着。当用户连上来时，此方法会返回一个Socket（在不同的 端口上）以便与客户端通信。Socket与ServerSocket 的端口不相同，因此ServerSocket可以空出来等待其他的用户。



DailyAdviceServer 程序代码

这个程序会创建ServerSocket并等待客户端的请求。当它收到客户端请求时，服务器会建立与客户端的Socket连接。服务器接着会建立PrintWriter来送出信息给客户端。

要记得import

这边的换行是因为排版的关系，千万别自己输入换行

嘉言来自这个数组

```
import java.io.*;
import java.net.*;

public class DailyAdviceServer {

   String[] adviceList = {"Take smaller bites", "Go for the tight jeans. No they do NOT
make you look fat.", "One word: inappropriate", "Just for today, be honest. Tell your
boss what you *really* think", "You might want to rethink that haircut."};

   public void go() {

      try {
         ServerSocket serverSock = new ServerSocket(4242);
```

ServerSocket会监听客户端对这台机器在4242端口上的要求

服务器进入无穷循环等待服务客户端的请求

这个方法会停下来等待要求到达之后才会继续

```
         while(true) {

            Socket sock = serverSock.accept();

            PrintWriter writer = new PrintWriter(sock.getOutputStream());
            String advice = getAdvice();
            writer.println(advice);
            writer.close();
            System.out.println(advice);
         }
```

使用Socket连接来送出嘉言信息，送出后就可以把连接关闭

```
      } catch(IOException ex) {
         ex.printStackTrace();
      }
   } // 关闭go函数

   private String getAdvice() {
      int random = (int) (Math.random() * adviceList.length);
      return adviceList[random];
   }

   public static void main(String[] args) {
      DailyAdviceServer server = new DailyAdviceServer();
      server.go();
   }
}
```

Brain Barbell

服务器如何知道怎样与客户端沟通?

客户端知道服务器的 IP 地址和端口号，但服务器是如何知道建立和客户端沟通的Socket连接的信息呢?

回想一下服务器是在何时何处如何取得客户端的信息。

there are no Dumb Questions

问： **上一页的嘉言服务器有个很严格的限制——看起来它每次只能服务一个用户!**

答： 没错! 在没有完成目前用户的响应程序循环之前它无法回到循环的开始处来处理下一个要求（无法进入 accept( ) 的等待来建立Socket 给新的用户）。

问： **换个方式再问一次，如何才能让服务器能够同时处理多个用户? 目前使用的方式根本应付不了聊天的需求。**

答： 这真的很简单，使用不同的线程并让新的客户端取得新的线程就好。我们正要开始讨论这个部分。

要点

- 客户端与服务器的应用程序通过Socket连接来沟通。
- Socket代表两个应用程序之间的连接，它们可能会是在不同的机器上执行的。
- 客户端必须知道服务器应用程序的IP地址（或网域名称）和端口号。
- TCP端口号是个16位的值，用来指定特定的应用程序。它能够让用户连接到服务器上各种不同的应用程序。
- 从0～1023的端口号是保留给HTTP、FTP、SMTP等已知的服务。
- 客户端通过建立Socket来连接服务器。

```
Socket s = new Socket("127.0.0.1", 4200);
```

- 一旦建立了连接，客户端可以从socket取得低层串流。

```
sock.getInputStream();
```

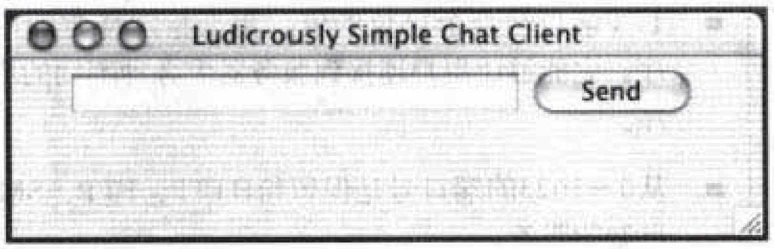
- 建立BufferedReader链接InputStreamReader与来自Socket的输入串流以读取服务器的文本数据。
- InputStreamReader是个转换字节成字符的桥梁。它主要是用来链接BufferedReader与低层的Socket输入串流。
- 建立直接链接Socket输出串流的PrintWriter请求print()方法 或println()方法来送出String给服务器。
- 服务器可以使用ServerSocket来等待用户对特定端口的请求。
- 当ServerSocket接到请求时，它会做一个Socket连接来接受客户端的请求。

编写聊天客户端程序

我们会用两阶段来编写聊天客户端应用程序。首先是只能发送信息给服务器，但不会收到其他来宾信息的版本（让聊天室整个概念废掉）。

然后我们会发展出可接收和发送信息的完整版本。

第一版：只能发送的版本



输入信息然后按下Send来传送给服务器。这个版本不会接收来自服务器的信息，所以也没有可滚动的文本区域

大致代码

```
public class SimpleChatClientA {

   JTextField outgoing;
   PrintWriter writer;
   Socket sock;

   public void go() {
        // 注册按钮的监听者
        // 调用setUpNetworking()
   }

   private void setUpNetworking() {

        // 建立Socket、PrintWriter

        // 赋值PrintWriter给实例变量
   }

   public class SendButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
          // 取得文字字段内容
          // 传送到服务器上
      }
   }  // 关闭SendButton  Listener内部类

} // 关闭外部类
```

```
import java.io.*;
import java.net.*;          记得要import
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClientA {

    JTextField outgoing;
    PrintWriter writer;
    Socket sock;

    public void go() {
        JFrame frame = new JFrame( "Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");                    建立GUI
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        setUpNetworking();
        frame.setSize(400,500);
        frame.setVisible(true);
    } // 关闭go                          使用localhost以便在同一
                                         台机器上作测试
    private void setUpNetworking() {
        try {
           sock = new Socket("127.0.0.1", 5000);                  建立Socket和PrintWriter
           writer = new PrintWriter(sock.getOutputStream());
           System.out.println( "networking established" );
        } catch(IOException ex) {
           ex.printStackTrace();
        }
    } // 关闭setUpNetworking

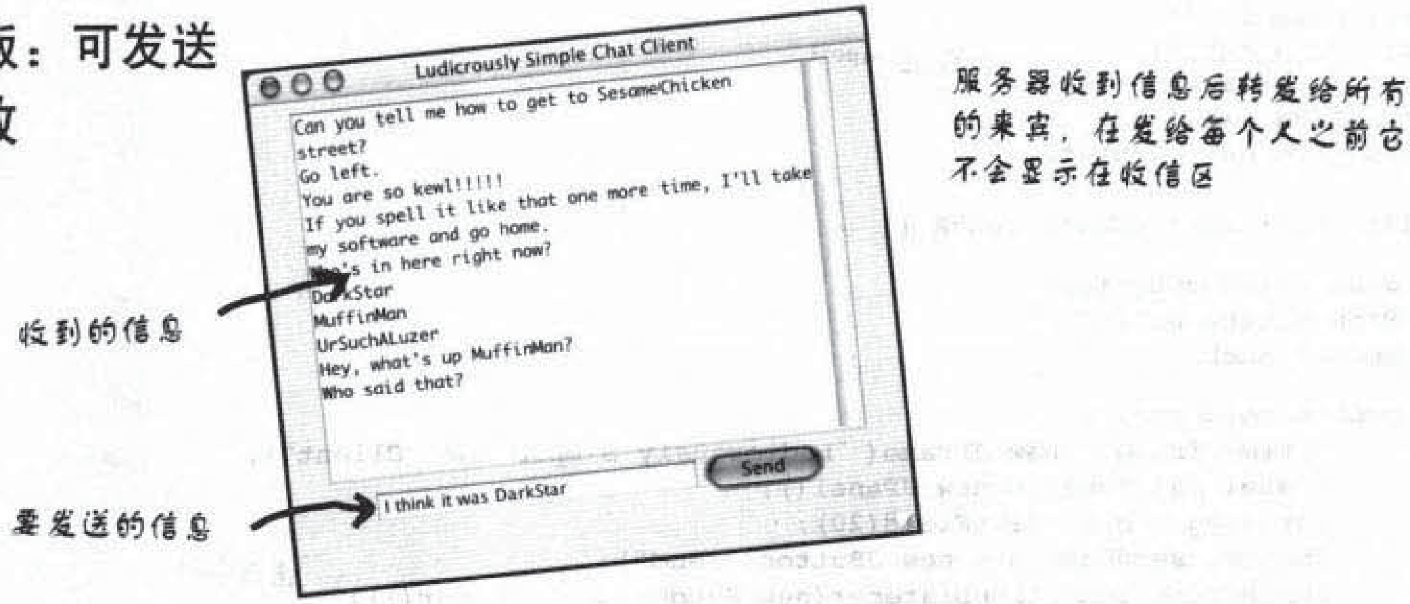
   public class SendButtonListener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
          try {
             writer.println(outgoing.getText());   可以开始写数据，所以println()会把信息
             writer.flush();                        送到服务器

          } catch(Exception ex) {
             ex.printStackTrace();
          }
          outgoing.setText("");
          outgoing.requestFocus();
      }
    }  // 关闭SendButtonListener内部类

   public static void main(String[] args) {
      new SimpleChatClientA().go();
   }
} // 关闭外部类
```

如果你现在要试试看的话，把本章后面列出来的程序打出来执行。先执行服务器然后再另外启动客户端。

第二版：可发送和接收



重要问题：如何从服务器取得信息？

应该很容易：当你设定网络的同时也把输入串流（或许是BufferedReader）建立好，然后使用readLine()读取信息。

重要问题：何时会从服务器取得信息？

想想看，有哪些选择？

① **选择一：每隔20秒查询服务器一次。**

优点：可行。

缺点：服务器要知道哪些是你已经看过的？哪些是你还没有看过的？这样一来服务器就得存储信息了。为什么是20秒？这样的延迟实在不好用，越是缩小延迟时间，又显得不太有效率。

② **选择二：每当用户送出信息时就顺便从服务器读回信息。**

优点：可行，简单。

缺点：蠢不可及。为什么要等到那个时候才检查信息。如果用户很懒都不传送信息怎么办？

③ **选择三：当信息被送到服务器上的时候就把它读回来。**

优点：高效率、最好用。

缺点：程序怎么写？这样得有个循环来等待服务器的信息。但要放在哪里？GUI 启动之后，除非有事件被触发，否则不会有任何一处是在运转的。



在Java中你可以边

咬口香糖边吹口哨

我们选择的是第三种方法

我们需要同时执行的能力，检查服务器信息的同时不会打断用户与GUI的交互！因此用户可以输入信息或滚动接收画面，还需要有东西在背景持续地读取服务器的数据。

这意味着我们需要新的线程（thread），一个独立的执行空间（stack）。

我们想要让只能发送的版本（第一版）可以维持原来的执行，同时有新的进程（process）并行地读取服务器的信息并将它显示在文本区域上。

除非你的计算机有多个处理器，否则Java上新的线程其实不会是运行在操作系统上独立的进程。但感觉上会像是那样。

Java的multithreading

Java在语言中就有内置多线程的功能。建立新的线程来执行是很简单的：

```
Thread t = new Thread();
t.start( );
```

就是这么的简单。建立出新的线程对象就会启动新的线程，它是个独立的调用执行空间。

但是会有一个问题。

该线程并没有执行任何程序。因此它几乎是一出生就变成植物人了。当线程脑死时，堆栈也就消失，故事就这么结束了。

因此，我们还少了一项关键因素——线程的任务，也就是独立线程要跑的程序代码。

Java的多个线程课题意味着我们得要讨论线程与它的任务，以及java.lang中的Thread这个类（要记得java.lang是默认就有被import的，它是包括String和System等语言本身的基础）。

Java有多个线程但只有一种Thread类

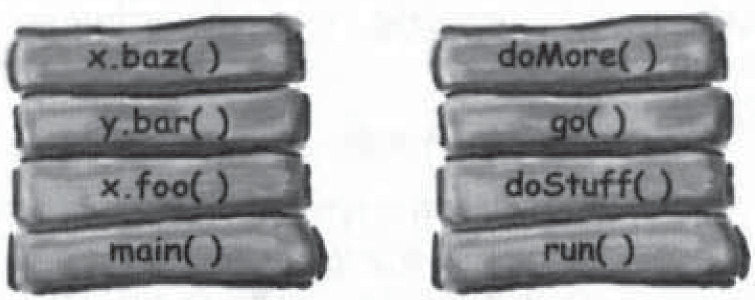
当你看到我们讨论线程时代表的是独立的线程，也就是独立的执行空间。当你看到Thread时，代表的是命名习惯，在Java中以大写字母开始的东西是类，所以Thread是java.lang这个包中的一个类。Thread对象代表线程，当你需要启动新的线程时就建立Thread的实例。

线程是独立的线程。它代表独立的执行空间。

Thread是Java中用来表示线程的类。

要建立线程就得创建Thread。

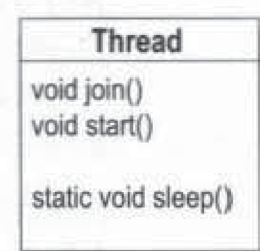
thread



主线程　　由程序启动的线程

线程是独立的线程，它代表独立的执行空间。每个Java应用程序会启动一个主线程——将main()放在它自己执行空间的最开始处。Java虚拟机会负责主线程的启动（以及比如垃圾收集所需的系统用线程）。程序员得负责启动自己建立的线程。

Thread



java.lang.Thread类

Thread是个表示线程的类。它有启动线程、连接线程和让线程闲置的方法（还有更多，这里只列出重要的几个）。

有一个以上的执行空间代表什么？

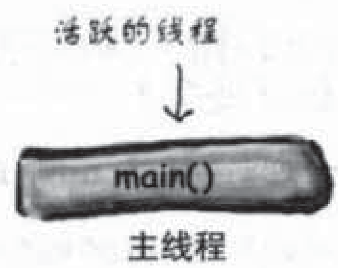
当有超过一个以上的执行空间时，看起来会像是有好几件事情同时发生。实际上，只有真正的多处理器系统能够同时执行好几件事，但使用Java的线程可以让它看起来好像同时都在执行中。也就是说，执行动作可以在执行空间非常快速地来回交换，因此你会感觉到每项任务都在执行。要记得，Java也只是个在低层操作系统上执行的进程。一旦轮到Java执行的时候，Java虚拟机实际上会执行什么？哪个字节码会被执行？答案是目前执行空间最上面的会被执行！在100个毫秒内，目前执行程序代码会被切换到不同空间上的不同方法。

线程要记录的一项事物是目前线程执行空间做到哪里。

它看起来会像下面这样：

1 Java虚拟机调用main()。

```
public static void main(String[] args) {
...
}
```

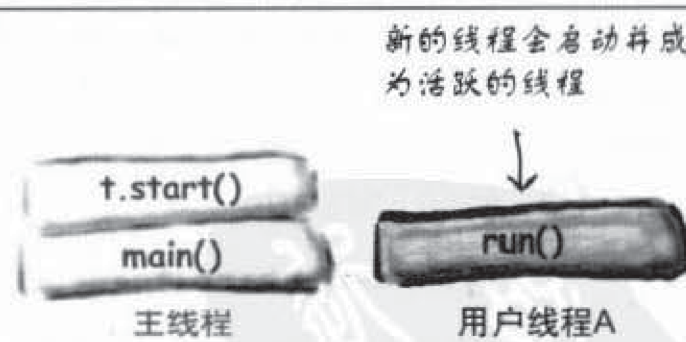


---

2 main( )启动新的线程。新的线程启动期间main的 线程会暂时停止执行。

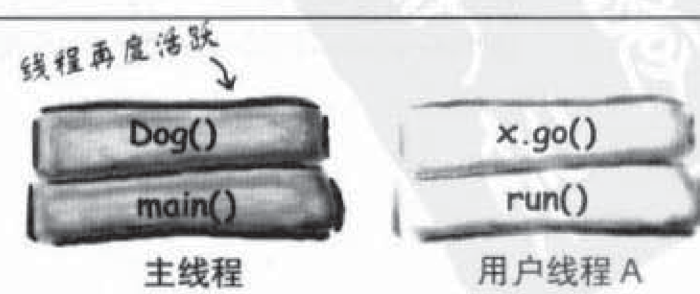
```
Runnable r = new MyThreadJob();
Thread t = new Thread(r);
t.start();
Dog d = new Dog();
```

稍后会说明



---

3 Java虚拟机会在线程与原来的主线程间切换直到两者都完成为止。



如何启动新的线程

❶ 建立Runnable对象（线程的任务）

```
Runnable threadJob = new MyRunnable();
```

稍后会对Runnable这个接口说明。你会编写实现Runnable的类，而此类就是你对线程要执行的任务的定义。也就是说此方法会在线程的执行空间运行。



❷ 建立Trhead对象（执行工人）并赋值Runnable（任务）

```
Thread myThread = new Thread(threadJob);
```

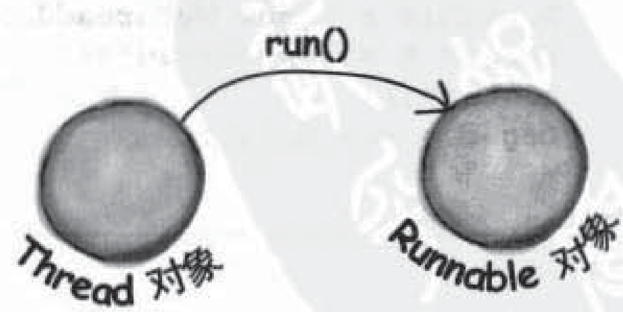
把Runnable对象传给Thread的构造函数。这会告诉Thread对象要把哪个方法放在执行空间去运行——Runnable的run()方法。



❸ 启动Thread

```
myThread.start();
```

在还没有调用Thread的start()方法之前什么也不会发生。这是你在只有一个Thread实例来建立新的线程时会发生的事情。当新的线程启动之后，它会把Runnable对象的方法摆到新的执行空间中。



每个Thread需要一个任务来执行。一个可以放在执行空间的任务。



对Thread而言，它是个工人，而Runnable就是这个工人的工作。

Runnable带有会放在执行空间的第一项方法：run( )。

Thread对象需要任务。任务是线程在启动时去执行的工作。该任务是新线程空间上的第一个方法，且它一定要长得像下面这样：

```
public void run() {
   // 会被新线程执行的代码
}
```

Runnable这个接口只有一个方法：public void run()。（要记得它是个接口，因此不管怎么写它都会是public的）

线程怎么会知道要先放上哪个方法？因为Runnable定义了一个协约。因为Runnable是个接口，线程的任务可以被定义在任何实现Runnable的类上。线程只在乎传入给Thread的构造函数的参数是否为实现Runnable的类。

当你把Runnable传给Thread的构造函数时，实际上就是在给Thread取得run( )的办法。这就等于你给了Thread一项任务。

实现Runnable接口来建立给thread运行的任务

它来自java.lang所以不需要import

```
public class MyRunnable implements Runnable {

    public void run() {
        go();
    }

    public void go() {
        doMore();
    }

    public void doMore() {
        System.out.println("top o' the stack");
    }
}
```

❷（标注 MyRunnable 类中的全部方法）

只有一个方法需要实现：public void run()，没有参数，要运行的程序放在这里

```
class ThreadTester {

    public static void main (String[] args) {

        Runnable threadJob = new MyRunnable();
        Thread myThread = new Thread(threadJob);

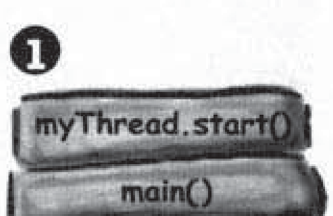
        myThread .start();

        System.out.println("back in main");
    }
}
```

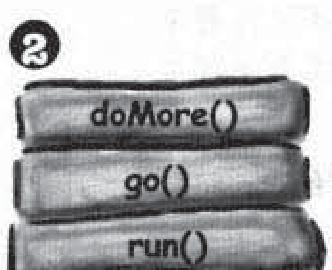
❶（标注 myThread .start();）

将Runnable的实例传给Thread的构造函数

要调用start()才会让线程开始执行。在此之前，它只是个Thread的实例，并不是真正的线程



主线程



新建线程

Brain Barbell

你想这个程序执行的结果会是什么？稍后会公布答案。

新建线程的3个状态




```
Thread t = new Thread(r);
```

Thread的实例已经创建，但还没有启动。也就是说，有Thread对象，没有执行中的线程。

```
t.start();
```

当你启动线程时，它会变成可执行的状态。意思是说它准备好要执行了，只要轮到它就可以开始。这时，该线程已经布置好执行空间。

所有的线程都在等待这一刻，成为执行中的那一个！这只能靠Java虚拟机的线程调度机制来决定。你有时也能对Java虚拟机选择执行线程给点意见，但无法强迫它把线程从可执行状态移动到执行中。

不只是这样。一旦线程进入可执行状态，它会在可执行与执行中两种状态中来来去去，同时也有另外一种状态：暂时不可执行（又称为被堵塞状态）。

典型的可执行/执行中循环
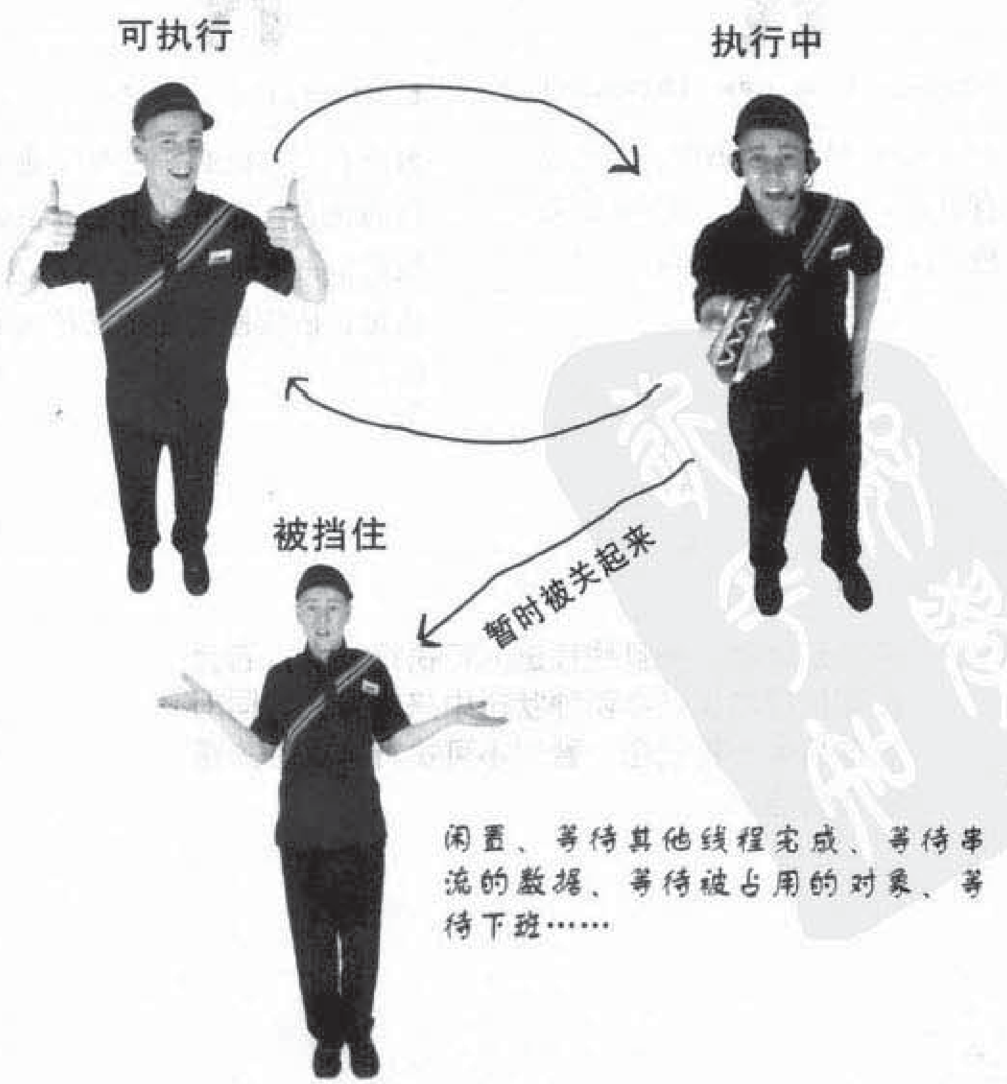
通常线程会在可执行与执行中两种状态中来回交替，因为Java虚拟机的线程调度会把线程挑出来运行又把它踢回去以让其他的线程有执行机会。



线程有可能会暂时被挡住

调度器（scheduler）会因为某些原因把线程送进去关一阵子。例如线程可能执行到等待Socket输入串流的程序段，但没有数据可供读取。调度器会把线程移出可执行状态，或者线程本身的程序会要求小睡一下（sleep()）。也有可能是因为线程调用某个被锁住（locked）的对象上的方法。此时线程就得等到锁住该对象的线程放开这个对象才能继续下去。

这类型的条件都会导致线程暂时失能。



线程调度器

线程调度器会决定哪个线程从等待状况中被挑出来运行，以及何时把哪个线程送回等待被执行的状态。它会决定某个线程要运行多久，当线程被踢出时，调度器也会指定线程要回去等待下一个机会或者是暂时地堵塞。

你无法控制调度，没有API可以调用调度器。最重要的是，调度无法确定（实际上可以做某种程度的保证，但是那也很模糊）。

至少不能让你的程序依靠调度的特定行为来保持执行的正确性！调度器在不同的Java虚拟机上面有不同的做法，就算同一个程序在同一台机器上运行也会有不同的遭遇。Java 程序设计新手会犯的最糟错误就是只在单一的机器上测试多线程程序，并假设其他机器的调度器都有相同的行为。

那写一次就可以到处跑的Java口号不是说很好玩的吗？它是表示你可以写与平台无关Java程序，不管线程调度器有怎样的行为，多线程程序一定可以运行。可是你不能假设每个线程都会被调度分配到公正平均的时间和顺序。且现今的Java虚拟机不太可能让你的线程一路执行到底。

原因在于sleep。没错，就是睡觉。让线程去睡个几毫秒才能让所有的线程都有机会被执行。线程的sleep()这个方法能够保证一件事：在指定的沉睡时间之前，昏睡中的线程一定不会被唤醒。举例来说，如果你要求线程去睡2000个毫秒，至少要等两秒过后它才会继续地执行。



线程调度器会做所有的决定。谁跑谁停都要看它。它通常是很公平的。但没有人能保证这件事，有时候某些线程很受宠，有些线程会被冷落。

显示调度器有多个不可预测的范例

找一台机器来运行这个程序

```
public class MyRunnable implements Runnable {

   public void run() {
      go();
   }

   public void go() {
      doMore();
   }

   public void doMore() {
      System.out.println("top o' the stack");
   }
}


class ThreadTestDrive {

   public static void main (String[] args) {

      Runnable threadJob = new MyRunnable();
      Thread myThread = new Thread(threadJob);

      myThread.start();

      System.out.println("back in main");
   }
}
```

产生这个输出：

```
File Edit Window Help PickMe
% java ThreadTestDrive
back in main
top o' the stack
% java ThreadTestDrive
top o' the stack
back in main
% java ThreadTestDrive
top o' the stack
back in main
% java ThreadTestDrive
top o' the stack
back in main
% java ThreadTestDrive
top o' the stack
back in main
% java ThreadTestDrive
top o' the stack
back in main
% java ThreadTestDrive
back in main
top o' the stack
```

注意到输出顺序是随机的，有时候主线程会先结束，有时候新建线程会先结束

PDG

怎么会有不同的结果?

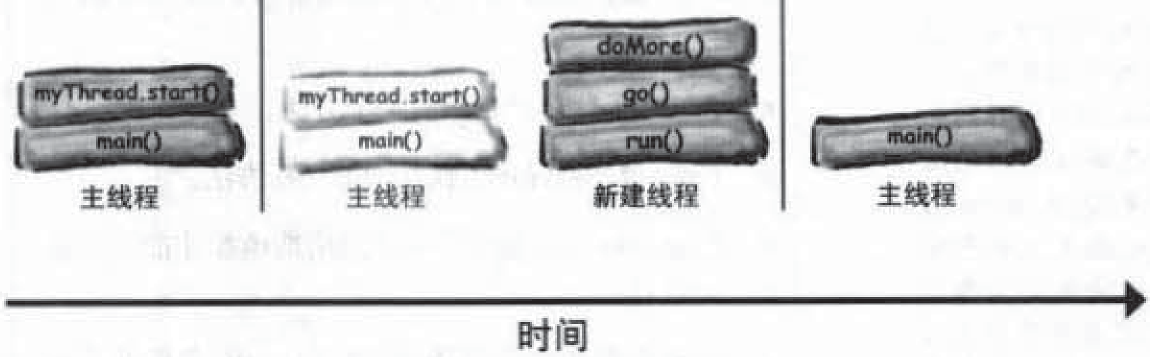
有时它会这样运行:

main()启动新的线程。

调度器把主线程搁置以便执行新的线程。

调度器让新的线程运行完打印出“top o..”。

新的线程结束，主线程恢复执行，列出“back..”。



有时却又是这样运行:

main()启动新的线程。

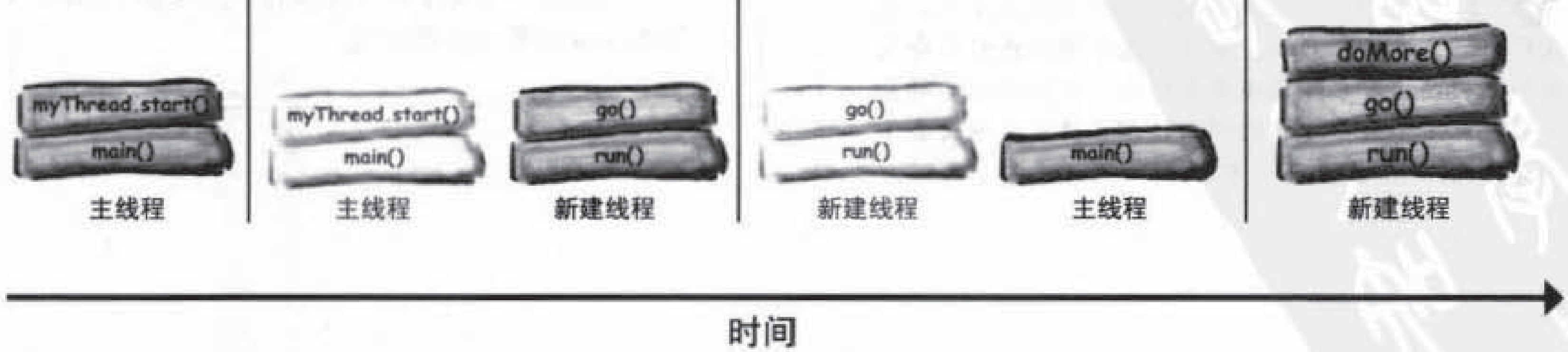
调度器把主线程搁置以便执行新的线程。

调度器执行一下新的线程就回到主线程。

调度器回到新的线程继续执行。

调度器再度回到主线程执行，列出信息。

最后新的线程才有机会执行到列出指令。



there are no Dumb Questions

问： 我曾经看过不使用Runnable的例子，而用Thread的子类来覆盖掉run()这个方法。然后调用Thread的无参数构造函数来创建出新的线程。

Thread t = new Thread( );

答： 是的，这是另外一种创建线程的方法，但却是以面向对象的观点来建立。子类的目的是什么？要记住我们现在讲的是两回事——Thread与线程的任务。从面向对象观点来看，这两者是非常不同的活动，也是不同的类。你唯一会想要子类/继承Thread的目的是要建立出更特殊的Thread。也就是说如果把Thread当作工人来看的话，除非有很特殊的工作行为，不然你不会继承Thread。如果你只是要有个工人来运行一般的任务，建立实现Runnable的独立类来给工人运行就行。

这跟设计概念有关，而不影响性能或语言用法的好坏。将Thread做个子类来覆盖掉run()是完全合法的，但通常不是个好主意。

问： Thread对象可以重复使用吗？能否调用start( )指定新的任务给它？

答： 不行。一旦线程的run()方法完成之后，该线程就不能再重新启动。事实上过了该点线程就会死翘翘。Thread对象可能还呆在堆上，如同活着的对象一般还能接受某些方法的调用，但已经永远地失去了线程的执行性，只剩下对象本身。

要点

- 以小写 t 描述的thread是个独立的线程。
- Java中的每个线程都有独立的执行空间。
- 大写T的Thread是java.lang.Thread这个类。它的对象是用来表示线程。
- Thread需要任务，任务是实现过Runnable的实例。
- Runnable这个接口只有一个方法。
- run( )会是新线程所执行的第一项方法。
- 要把Runnable传给Thread的构造函数才能启动新的线程。
- 线程在初始化以后还没有调用start()之前处于新建立的状态。
- 调用Thread对象的start()之后，会建立出新的执行空间，它处于可执行状态等待被挑出来执行。
- 当Java虚拟机的调度器选择某个线程之后它就处于执行中的状态，单处理器的机器只能有一个执行中的 线程。
- 有时线程会因为某些原因而被堵塞。
- 调度不能保证任何的执行时间和顺序，所以你不能期待它会完全地平均分配执行，你最多也只能影响sleep的最小保证时间。

让线程小睡一下



确保线程能够有机会执行的最好方式是让它们周期性地去睡一下*。你只要调用sleep()这个方法，传入以毫秒指定的时间就行。

```
Thread.sleep(2000);
```

这会把线程敲昏，而保持两秒之内不会醒来进入可执行状态。

但是这个方法有可能会抛出InterruptedException异常，所有对它的调用都必须包在try/catch块中。因此真正的程序代码会像是这样：

```
try {
    Thread.sleep(2000);
} catch(InterruptedException ex) {
    ex.printStackTrace( );
}
```

你写的线程或许永远也不会被中断，这个异常是API用来支持线程间通信的机制，实际上几乎没有人这样做。但良好的习惯规则会要求我们把有可能抛出异常的调用做妥善的处理。

现在你能够确定在指定时间内不会醒来，但是线程也不一定会在时间过后马上醒来直接变成执行中的状态，你只能确定它会回到可执行的状态。何时执行还是要看调用器大哥的意思。这样的运行对时间控制来说不是非常的准确，但在一般机器上没有太多线程在运行的时候还过得去。千万不要依靠这种机制来精确地控制执行时机（例如说画面与声音的同步，不然就有可能在张嘴之前就听到说话，挺恐怖的）。

如果想要确保其他的线程有机会执行的话，就把线程放进睡眠状态。

当线程醒来的时候，它会进入可执行状态等待被调度器挑出来执行。

*译注：这一页关于sleep()的说法虽然不能算错，但概念上可能会引起误解。我们通常不必刻意使用sleep()来保证其他的线程会被执行。这个问题很复杂，需要一整本书才能讲清楚。

使用 sleep() 让程序更加可预测

之前不是有个范例显示出每次运行都可能有不同的结果吗？回头看一下它的程序与输出。有时主程序必须要等到线程做完，有时主程序却会先做完。我们要怎样修正这个状况？先停下来想一下这个问题："在哪边加上sleep()可以让back……在top……之前打印出？"。

等你想出一个答案再继续下去（可行的答案有好几种）。

有了吗？（恭喜！）

我们想要这样的结果：一致的输出顺序：

```
File Edit Window Help SnoozeButton
% java ThreadTestDrive
back in main
top o' the stack
% java ThreadTestDrive
back in main
top o' the stack
% java ThreadTestDrive
back in main
top o' the stack
% java ThreadTestDrive
back in main
top o' the stack
% java ThreadTestDrive
back in main
top o' the stack
```

```
public class MyRunnable implements Runnable {

   public void run() {
      go();
   }

   public void go() {

      try {
        Thread.sleep(2000);
      } catch(InterruptedException ex) {
        ex.printStackTrace();
      }

      doMore();
   }

   public void doMore() {
      System.out.println("top o' the stack");
   }
}

class ThreadTestDrive {
   public static void main (String[] args) {
      Runnable theJob = new MyRunnable();
      Thread t = new Thread(theJob);
      t.start();
      System.out.println("back in main");
   }
}
```

调用sleep()来强迫此线程离开执行中的状态

主线程会变成执行中的状态，做完之后再度回到新线程继续执行，因此打印输出的顺序就会符合我们的预期，但两次输出之间会有约两秒的空白……

译注：又是我。实在不想自嘲，但是这个例子不是很好。这种方式只会浪费两秒的时间，也不是能够100%保证顺序一致。想象一下如果调用sleep()之后操作系统刚好去读磁盘驱动器而花了两秒，回头会执行哪一个线程可就不一定了！（都设成200秒会不会好一点呢？）

建立与启动两个线程

线程可以有名字。你可以找老师帮线程取个好听又能够走运的好名字，或是使用默认的名称。但最酷的事情还是你可以用名字来判别正在运行的是哪个线程。下面的例子有两个线程。它们都执行相同的工作：在循环中列出线程的名称。

```
public class RunThreads implements Runnable {

    public static void main(String[] args) {
        RunThreads runner = new RunThreads();
        Thread alpha = new Thread(runner);
        Thread beta = new Thread(runner);
        alpha.setName("Alpha thread");
        beta.setName("Beta thread");
        alpha.start();
        beta.start();
    }

    public void run() {
        for (int i = 0; i < 25; i++) {
            String threadName = Thread.currentThread().getName();
            System.out.println(threadName + " is running");
        }
    }
}
```

创建Runnable的实例

创建两个线程，使用相同的Runnable（相同的任务——稍后会讨论这种做法）

帮线程取名字

启动线程

两个线程都会运行这个循环把名字列出来

会发生什么事？

部分的输出

```
File Edit Window Help Centauri
Alpha thread is running
Alpha thread is running
Alpha thread is running
Beta thread is running
Alpha thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Beta thread is running
Alpha thread is running
```

线程会轮流执行吗？会交互出现输出吗？多久切换一次？每做完一圈就交换吗？还是要5圈后才交换？

你已经知道答案了：不知道！这都只能听命于调度器的安排。根据操作系统、使用的Java虚拟机版本、CPU等，你会运行出不一样的结果。

在OS X 10.2上面以5圈或更少的圈数来运行时，Alpha会先做完，然后Beta才会做完。结果很一致，然而不保证永远都是这样。

但如果跑上25圈时就不太一样了，Alpha可能跑不完25圈就会切换到Beta上面。



呃……实际上有缺点的。线程会产生并发性的问题。

并发性（concurrency）问题会引发竞争状态（race condition）。竞争状态会引发数据的损毁。数据损毁会引发恐惧……最后连花儿都不开、鸟儿都不叫、人们的脸上也失去了笑容。

这一切都来自于可能发生的一种状况：两个或以上的线程存取单一对象的数据。也就是说两个不同执行空间上的方法都在堆上对同一个对象执行getter或setter。

两个线程各自认为自己是宇宙的中心，只关心自己的任务。因为线程会被打入可执行状态，此时基本上是昏迷过去的，当它回到执行中的状态时，根本也不知道自己曾经不省人事。

婚姻亮起了红灯。这一对怨偶有救吗？



接下来为您播出

“鬼话连篇之马丁博士谈婚姻”

[第 42 集的对白]

欢迎收看鬼话连篇。

我们今天讲的是关于夫妻不和最主要的两个原因：金钱和睡眠。



怨偶天成：杰纶与沛晨

杰纶与沛晨这一对冤家同居并且也把钱存在一起。但如果不把问题解决掉的话很快就会分手。什么问题呢？典型的贫贱夫妻百事哀。

沛晨是这么抱怨的：

“杰纶跟我说好了不会透支花费，所以规定每个人花钱之前必须先检查余额，这看起来很简单，但有一天我们就突然发现连预借现金的额度也用掉了。

我做梦也没想到会有这么一天，事情是这样的：

杰纶需要 5 万元上夜店，所以查了一下可用额度还有10万元，是可以把钱提出来的。但他没有先提钱，反而就不知不觉地睡了起来。



杰纶有嗜睡的问题，他会在检查余额与提款的中途睡着，如果一睡不起就算了，可是他总是会醒来领钱，却没有再检查一次余额。

我回家时他还在睡，而我想买个价值10万元的新包包，因此查了一下还有10万元的额度（杰纶还没有提走钱，所以额度不变），真是太幸运了。然后我把钱提出来去买了包包。接着杰纶也醒了，他就去提钱上夜店。我们就是这样开始产生问题！他根本不知道自己睡了多久，所以醒来之后就去领钱，而没有再检查一次账户余额。

马丁博士，我该怎么办？”

有办法解决吗？他们这一对是不是就毁了？我们无法停止杰纶昏睡的毛病，但是我们能不能要求沛晨在杰纶睡着的时候不要碰提款卡和存折？

不要走开，广告之后我们马上回来看他们的笑话……呃，对不起，是悲剧。没关系，别回来，我们马上走开！

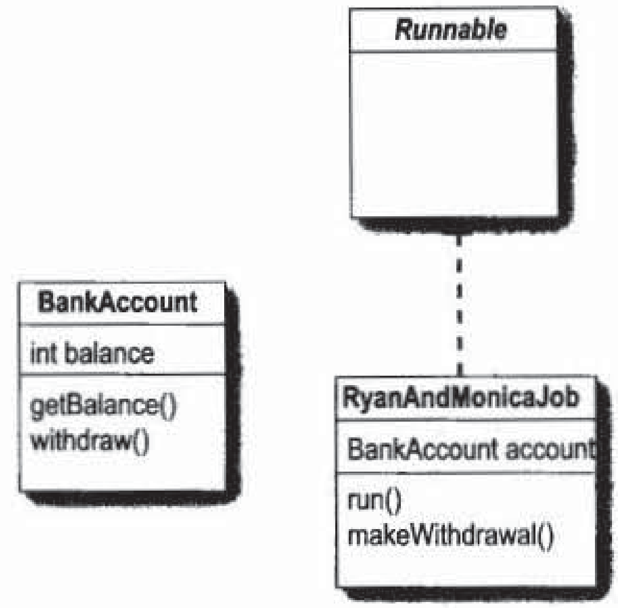
以程序码表示杰纶与沛晨的问题

以下的范例展示出两个线程（杰纶 = Ryan，沛晨 = Monica）间共享单一对象（银行账户）时可能会出什么状况。

这个程序代码有两个类，BankAccount与RyanAndMonicaJob。RyanAndMonicaJob这个类实现Runnable，用来代表两人都有的行为——检查余额然后花掉。当然啦，每个线程都会在检查余额与实际提款之间偷睡一下。

MonicaAndRyanJob这个类有个类型为BankAccount的实例变量，它代表共享的账户。

程序代码的工作方式是这样：



1 建立一个RyanAndMonicaJob的实例

它是个Runnable的类（要被执行的任务），因为两人的行为都一样，所以我们只需要一个实例。

```
RyanAndMonicaJob  theJob = new RyanAndMonicaJob();
```

2 以同一个Runnable建立两个线程（RyanAndMonicaJob 的实例）

```
Thread one = new Thread(theJob);
Thread two = new Thread(theJob);
```

3 命名并启动线程

```
one.setName("Ryan");
two.setName("Monica");
one.start();
two.start();
```

4 观察两个线程执行run()

两个线程分别代表两人。它们都会持续地检查账户，然后只会在余额足够的情况下提款！

```
if (account.getBalance() >= amount) {
   try {
      Thread.sleep(500);
   } catch(InterruptedException ex) {ex.printStackTrace(); }
}
```

在run()这个方法中，它会执行跟故事所说一样的操作。

理论上这应该不会遇到提款过多的问题。

除非……杰纶和沛晨都会在检查账户与提款之间睡着。

杰伦与沛晨的范例

```
class BankAccount {
   private int balance = 100;

   public int getBalance() {
      return balance;
   }
   public void withdraw(int amount) {
      balance = balance - amount;
   }
}
```

账户一开始有100元

```
public class RyanAndMonicaJob implements Runnable {

   private BankAccount account = new BankAccount();
```

只有一个RyanAndMonicaJob的实例，代表只有一个共享的账户

```
   public static void main (String [] args) {
      RyanAndMonicaJob  theJob = new RyanAndMonicaJob();
      Thread one = new Thread(theJob);
      Thread two = new Thread(theJob);
      one.setName("Ryan");
      two.setName("Monica");
      one.start();
      two.start();
    }
```

将任务初始化

创建出使用相同任务的两个线程，这代表两个线程都会存取同一个账户

```
  public void run() {
   for (int x = 0; x < 10; x++) {
      makeWithdrawal(10);
      if (account.getBalance() < 0) {
        System.out.println("Overdrawn!");
      }
    }
 }
```

检查账户余额，如果透支就列出信息，不然就去睡一会儿，然后醒来完成提款操作

```
  private void makeWithdrawal(int amount) {
     if (account.getBalance() >= amount) {
     System.out.println(Thread.currentThread().getName() + " is about to withdraw");
      try {
        System.out.println(Thread.currentThread().getName() + " is going to sleep");
        Thread.sleep(500);
      } catch(InterruptedException ex) {ex.printStackTrace(); }
      System.out.println(Thread.currentThread().getName() + " woke up.");
      account.withdraw(amount);
      System.out.println(Thread.currentThread().getName() + " completes the withdrawl");
      }
    else {
      System.out.println("Sorry, not enough for " + Thread.currentThread().getName());
    }
  }
}
```

检查账户余额，如果透支就列出信息，不然就去睡一会儿，然后醒来完成提款操作

列出信息以便执行时进行观察

File Edit Window Help Visa

```
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Monica is about to withdraw
Monica is going to sleep
Ryan woke up.
Ryan completes the withdrawl
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Monica is about to withdraw
Monica is going to sleep
Ryan woke up.
Ryan completes the withdrawl
Ryan is about to withdraw
Ryan is going to sleep
Monica woke up.
Monica completes the withdrawl
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Sorry, not enough for Monica
Ryan woke up.
Ryan completes the withdrawl
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
Sorry, not enough for Ryan
Overdrawn!
```

怎么会发生这状况？

makeWithdrawal()这个方法会在提款之前检查账户余额，但不该发生的事情还是发生了。

以下是这个状况的说明：

杰纶检查余额，看到钱够，然后去睡。

同时间，沛晨也来检查余额，也看到有足够的余额。它不知道杰纶等下醒来就要去提款。

沛晨睡了，还流了几滴口水……

杰纶醒来，抽了一根烟，然后把钱提走。

沛晨醒来，发现口水发臭，赶忙洗了把脸，然后提款。这下问题可大了！

沛晨检查余额发现已经透支了，她先是惊慌，然后转为愤怒，最后发现这辈子第一次有想要抢银行的冲动。

沛晨必须要学习在杰纶还没有醒来完成提款之前不能去查询账户余额。反之亦然。

他们需要对账户存取的一道锁

这锁会像这样工作：

① 锁会与银行账户交易有关（检查余额与提款）。只有一把钥匙，且它会与锁摆在一起直到有人想要存取账户为止。



没有交易时，锁是开着的。

② 当杰纶要存取银行账户时，他会把账户锁上并收起钥匙。现在就没有其他人能够存取账户了。



杰纶想要交易，所以把账户给锁上并带走钥匙。

③ 杰纶会持有钥匙直到完成交易为止。唯一的钥匙在杰纶手上，因此沛晨无法存取账户，除非杰纶解锁并放回钥匙。

现在就算杰纶检查完账户就去睡觉，也能够保证醒来时账户还是维持原状，因为只有他才有钥匙。



交易完成后就解锁并归还钥匙。现在就可以换其他人存取账户。

我们需要让makeWithdrawal()跑起来像个原子





我们必须要确定线程一旦进入makeWithdrawal()这个方法之后，它就必须要能够在其他线程进入之前把任务执行完毕。

也就是说我们需要确保线程一旦开始检查账户余额，就要能够确定它会在任何其他线程检查账户余额之前醒来把提款动作完成。

使用**synchronized**这个关键词来修饰方法使它每次只能被单一的线程存取。

这就是保护银行账户的方法！你不会把锁上到账户本身，但你会锁上执行银行交易的方法。如此一来，线程会从头到尾完成交易，就算中途昏睡过去也一样！

如果没有锁上账户，那到底锁了什么？是方法吗？还是Runnable对象？难道是线程本身？

答案在下一页揭晓。然而程序本身是很简单的——只要把synchronized这个修饰符加到方法的声明上就可以。

synchronized关键词代表线程需要一把钥匙来存取被同步化（synchronized）过的线程。

要保护数据，就把作用在数据上的方法给同步化。

```
private synchronized void makeWithdrawal(int amount) {

    if (account.getBalance() >= amount) {
       System.out.println(Thread.currentThread().getName() + " is about to withdraw");
       try {
          System.out.println(Thread.currentThread().getName() + " is going to sleep");
          Thread.sleep(500);
       } catch(InterruptedException ex) {ex.printStackTrace(); }
       System.out.println(Thread.currentThread().getName() + " woke up.");
       account.withdraw(amount);
       System.out.println(Thread.currentThread().getName() + " completes the withdrawl");
   } else {
      System.out.println("Sorry, not enough for " + Thread.currentThread().getName());
   }
}
```



（物理系的同学请注意：原文所说的原子是"atomic"，这是一种"古典"的说法，古人认为原子是不可分割的最小物质单位，拿来描述我们所说的不可切割工作是很方便的，但这种用法不是我们发明的，不然我们会引用海森堡测不准原理来卖弄一下我们在现代量子物理学上的功力。）

使用对象的锁

每个对象都有个锁。大部分时间都没有锁上，并且你可以假设有个虚拟的钥匙随侍在旁。对象的锁只会在有同步化的方法上起作用。当对象有一个或多个同步化的方法时，线程只有在取得对象锁的钥匙时才能进入同步化的方法。



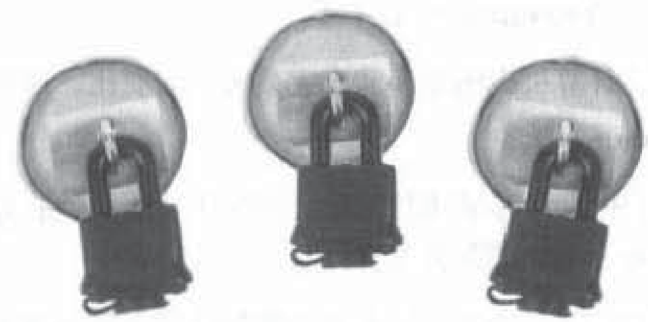
锁不是配在方法上的，而是配在对象上。如果对象有两个同步化的方法，就表示两个线程无法进入同一个方法，也表示两个线程无法进入不同的方法。

想想看，如果你有多个方法可能会操作对象的实例变量，则这些方法都应该要有同步化的保护。

同步化的目标是要保护重要的数据。但要记住，你锁住的不是数据而是存取数据的方法。

所以线程在开始执行并遇上有同步化的方法时候会发生什么事？线程会认知到它需要对象的钥匙才能进入该方法。它会取得钥匙（这是由Java虚拟机来处理，没有存取对象锁的API可用），如果可以拿到钥匙才会进入方法。

从这一点开始，线程会全力照顾好这个钥匙，除非完成同步化的方法，否则它不会放开钥匙。因此当线程持有钥匙时，没有其他的线程可以进入该对象的同步化方法，因为每个对象只有一个钥匙。



每个Java对象都有一个锁，每个锁只有一把钥匙。

通常对象都没上锁，也没有人在乎这件事。

但如果对象有同步化的方法，则线程只能在取得钥匙的情况下进入线程。也就是说并没有其他线程已经进入的情况下才能进入。

“丢失更新”问题令人闻风丧胆

下面是另一个典型的并行性（concurrency）问题，这是数据库领域的说法。它跟杰纶与沛晨这个悲剧结合的故事类似，但我们另外使用下面这个例子来展示出几项重点。

丢失更新（lost update）有一种特定的过程。

（1）取得账户余额。

```
int i = balance;
```

（2）将账户余额加1。

```
balance = i + 1;
```

这会让计算机以两个步骤来完成账户的变化。通常你会以单一的命令来做这件事：

```
balance++;
```

但强行以两个步骤来处理就会浮现出非原子性的问题。因此你可以想象到如果步骤复杂到无法以单一命令来完成时会怎样。

下面我们用两个都想把余额递增的线程来展示丢失更新。

```
class TestSync implements Runnable {

    private int balance;

    public void run() {
       for(int i = 0; i < 50; i++) {
         increment();
         System.out.println("balance is " + balance);
       }
    }

    public void increment() {
       int i = balance;
       balance = i + 1;
    }
}
```

每个线程都把账户递增50次

问题出在这里，我们用的是读取时的值而不是目前的值

```
public class TestSyncTest {
   public static void main (String[] args) {
     TestSync job = new TestSync();
     Thread a = new Thread(job);
     Thread b = new Thread(job);
     a.start();
     b.start();
   }
}
```

执行程序

① thread A 运行了几下



把账户余额读进变量 i；账户为 0，所以 i 为 0。

设定账户余额为i + 1；账户余额变成 1。

把账户余额读进变量i；账户为1，所以i为 1。

设定账户余额为i + 1；账户余额变成2。

② 换thread B运行几下



把账户余额读进变量i；账户为2，所以i为2。

设定账户余额为i + 1；账户余额变成3。

把账户余额读进变量i；账户为3，所以i为3。

然后B睡着了，还没有把账户余额设为4。

③ 再换thread A从上次做完的位置开始运行几下。



把账户余额读进变量i；账户为3，所以i为3。

设定账户余额为i + 1；账户余额变成 4。

把账户余额读进变量i；账户为4，所以i为 4。

设定账户余额为i + 1；账户余额变成5。

④ 轮到thread B运行。



设定账户余额为i + 1；账户余额变成4。

哎呀！

B把A所做过的动作覆盖掉，让A的更新看起来好像从未发生一般

thread A的更新就这么消失了！

thread B之前读取过数据，等到醒来后，它就继续写入的操作，完全不知道中间丧失过意识

用同步机制让 increment() 方法原子化！

将increment()方法同步化可以解决丢失更新问题，因为它会让方法中的两个步骤组成不可分割的单元。

```
public synchronized void increment() {
    int i = balance;
    balance = i + 1;
}
```

一旦线程进入了方法，我们必须确保在其他线程可以进入该方法之前所有的步骤都会完成（如同原子不可分割一样）。

---

there are no Dumb Questions

问： **听起来把所有东西都同步化是个不错的主意，如此一来全部都会具有多线程执行的安全性。**

答： 馊主意。同步化不是没有代价的。首先，同步化的方法有些额外的成本。也就是说进入同步化方法的程序会查询钥匙等性能上的损耗（虽然你不太会注意到）。

其次，同步化的方法会让你的程序因为要同步并行的问题而慢下来。换句话说，同步化会强制线程排队等着执行方法，也许听起来没什么，但你得要想想一开始为什么要写多线程并行的程序。

最后，最可怕的是同步化可能会导致死锁现象（见516页）。

原则上最好只做最少量的同步化。事实上同步化的规模可以小于方法全部。本书不会深入这个部分，但你可用 synchronized 来修饰一行或数行的指令而不必整个方法都同步化。

不需要整个都同步化

```
public void go() {
  doStuff();

  synchronized(this) {
     criticalStuff();
     moreCriticalStuff();
  }
}
```

只有这两个调用要被组合成原子单位。同步化的这种用法不会将整个方法设定成需要同步化，只会使用参数所指定的对象的锁来做同步化。

虽然有别的方法可以达到同样的效果，但你通常会以当前对象（this）来同步化。

① thread A运行了几下



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

把账户余额读进变量i；账户为0，所以i为0。

设定账户余额为i + 1；账户余额变成1。

归还钥匙。

再重新进入，取得钥匙，余额为1，所以i为1。

[进入睡眠状态，但因为还没有完成同步化的方法，所以钥匙还在手上]

② 换thread B运行几下



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

拿不到钥匙。

[B只好进入等待对象锁钥的状态]

③ 再换thread A从上次做完的位置开始运行几下（注意到钥匙还在手上）

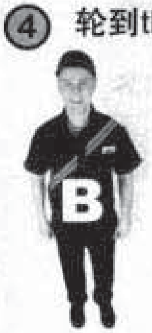


设定账户余额为i + 1；账户余额变成2。

归还钥匙。

[A进入可执行状态]

④ 轮到thread B运行



尝试进入increment()方法，因为是同步化的方法，所以要取得钥匙。

这一次拿到钥匙了……

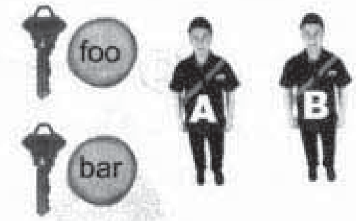
同步化的死亡阴影

使用同步化的程序代码要小心，因为没有其他的东西能够像线程的死锁（deadlock）这样伤害你的程序。死锁会发生是因为两个线程互相持有对方正在等待的东西。没有方法可以脱离这个情况，所以两个线程只好停下来等，一直等，一直等，海枯石烂还在继续等。

如果你对数据库或其他的应用程序服务器很熟，那你就应该知道这个问题：数据库有与同步化非常相似的上锁机制。但像样的数据库交易管理系统有时能处理掉死锁。例如它可能会把等待太久的交易视为死锁。但与Java不同的地方在于它们有事务回滚机制来复原不能全部完成的交易。

Java没有处理死锁的机制。它甚至不会知道死锁的发生。所以你得小心设计程序。如果你经常编写多线程的程序，建议阅读O'Reilly出版的“Java Thread”，上面有一些观念的澄清和设计的提示可以帮忙避免死锁（译注：有中文译本，译文精确优雅，译者帅气逼人，是本不可多得的好书）。

只要两个线程和两个对象就可以引起死锁





要点

- Thread.sleep()这个静态方法可以强制线程进入等待状态到过了设定时间为止，例如Thread.sleep(200)会睡上200个毫秒。
- 可以调用sleep()让所有的线程都有机会运行。
- sleep()方法可能会抛出InterrruptedException异常，所以要包在try/catch块，或者把它也声明出来。
- 你可以用setName()方法来帮线程命名，通常是用来除错的。
- 如果两个或以上的线程存取堆上相同的对象可能会出现严重的问题。
- 如果两个或两个以上的线程存取相同的对象可能会引发数据的损毁。
- 要让对象在线程上有足够的安全性，就要判断出哪些指令不能被分割执行。
- 使用**synchronized**这个关键词修饰符可以防止两个线程同时进入同一对象的同一方法。
- 每个对象都有单一的锁，单一的钥匙。这只会在对象带有同步化 方法时才有实际的用途。
- 线程尝试要进入同步化过的方法时必须要取得对象的钥匙，如果因为已经被别的线程拿走了，那就得等。
- 对象就算是有多个同步化过的方法，也还是只有一个锁。一旦某个线程进入该对象的同步化方法，其他线程就无法进入该对象上的任何同步化线程。

全新配方的 SimpleChatClient

回到本章开始的主题，我们创建出SimpleChatClinet来发送信息给服务器，但无法接收。这就是为什么要讨论线程的原因，因为我们需要能够同时做两件事的方法：送出信息给服务器的同时读取来自服务器的信息，并显示在可滚动的区域。

是的，本章要结束了，但是……

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SimpleChatClient {

    JTextArea incoming;
    JTextField outgoing;
    BufferedReader reader;
    PrintWriter writer;
    Socket sock;

    public static void main(String[] args) {
       SimpleChatClient client = new SimpleChatClient();
       client.go();
    }

    public void go() {

        JFrame frame = new JFrame("Ludicrously Simple Chat Client");
        JPanel mainPanel = new JPanel();
        incoming = new JTextArea(15,50);
        incoming.setLineWrap(true);
        incoming.setWrapStyleWord(true);
        incoming.setEditable(false);
        JScrollPane qScroller = new JScrollPane(incoming);
        qScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
        qScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        outgoing = new JTextField(20);
        JButton sendButton = new JButton("Send");
        sendButton.addActionListener(new SendButtonListener());
        mainPanel.add(qScroller);
        mainPanel.add(outgoing);
        mainPanel.add(sendButton);
        setUpNetworking();

        Thread readerThread = new Thread(new IncomingReader());
        readerThread.start();

        frame.getContentPane().add(BorderLayout.CENTER, mainPanel);
        frame.setSize(400,500);
        frame.setVisible(true);

    } // 关闭go
```

除了标记出来的段落之外，这边都是之前看过的GUI程序代码

启动新的线程，以内部类作为任务，此任务是读取服务器的socket串流显示在来文本区域

```
private void setUpNetworking() {

     try {
        sock = new Socket("127.0.0.1", 5000);
        InputStreamReader streamReader = new InputStreamReader(sock.getInputStream());
        reader = new BufferedReader(streamReader);
        writer = new PrintWriter(sock.getOutputStream());
        System.out.println("networking established");
     } catch(IOException ex) {
        ex.printStackTrace();
     }
  } // 关闭setUpNetworking
```

使用socket取得输入/输出的串流

```
public class SendButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
       try {
          writer.println(outgoing.getText());
          writer.flush();

       } catch(Exception ex) {
          ex.printStackTrace();
       }
       outgoing.setText("");
       outgoing.requestFocus();
   }
 } // 关闭内部类
```

用户按下send按钮时送出文本字段的内容到服务器上

```
public class IncomingReader implements Runnable {
   public void run() {
        String message;
        try {

          while ((message = reader.readLine()) != null) {
             System.out.println("read " + message);
             incoming.append(message + "\n");

           } // while结束
        } catch(Exception ex) {ex.printStackTrace();}
     } // 关闭run()
  } // 关闭内部类

} // 关闭外部类
```

thread的任务！

run()会持续地读取服务器信息并把它加到可滚动的文本区域上



现成码

非常非常简单的聊天服务器程序

你可以用这个服务器程序来服务先后两版的聊天客户端程序。我们去掉很多功能来让它精简。也就是说，这个程序可用，但至少有100种方法可以让它死掉。如果你真的很想磨练你的功夫，可以在这本书结束之后回头加强这个程序。

另外一种现在就可以进行锻炼的方法是自己帮程序加注释。如果你自己搞定程序的来龙去脉会比我们跟你说要好。再说一次，这边列出的是现成可用的程序，所以不一定要全部看懂。它的目的只是为了要让客户端可以测试。

你需要用一个命令列来启动服务器，然后再用另外一个命令列来启动客户端。

```java
import java.io.*;
import java.net.*;
import java.util.*;

public class VerySimpleChatServer {

    ArrayList clientOutputStreams;

    public class ClientHandler implements Runnable {
         BufferedReader reader;
         Socket sock;

         public ClientHandler(Socket clientSocket) {
           try {
             sock = clientSocket;
             InputStreamReader isReader = new InputStreamReader(sock.getInputStream());
             reader = new BufferedReader(isReader);

           } catch(Exception ex) {ex.printStackTrace();}
          } // 关闭构造函数

        public void run() {
           String message;
           try {
             while ((message = reader.readLine()) != null) {
                System.out.println("read " + message);
                tellEveryone(message);

              } // 结束while循环
           } catch(Exception ex) {ex.printStackTrace();}
       } // 关闭run()
   } // 关闭内部类
```

```
    public static void main (String[] args) {
         new VerySimpleChatServer().go();
    }

    public void go() {
       clientOutputStreams = new ArrayList();
       try {
         ServerSocket serverSock = new ServerSocket(5000);

         while(true) {
            Socket clientSocket = serverSock.accept();
            PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());
            clientOutputStreams.add(writer);

            Thread t = new Thread(new ClientHandler(clientSocket));
            t.start();
            System.out.println("got a connection");
         }

       } catch(Exception ex) {
         ex.printStackTrace();
       }
   } // 关闭go

   public void tellEveryone(String message) {

      Iterator it = clientOutputStreams.iterator();
      while(it.hasNext()) {
         try {
            PrintWriter writer = (PrintWriter) it.next();
            writer.println(message);
            writer.flush();
         } catch(Exception ex) {
              ex.printStackTrace();
         }

      } // 结束while

   } // 关闭tellEveryone
} // 关闭类
```

there are no Dumb Questions

问：那么静态变量状态的保护呢？如果有静态的方法可以对静态变量的状态作更新，还能够用同步化吗？

答： 可以！记得静态的方法是运行在类而不是每个实例上的吗？所以你可能会猜想要用哪个对象的锁。毕竟有可能完全没有该类的实例存在。幸好对象有锁，每个被载入的类也有个锁。这表示说如果有3个Dog对象在堆上，则总共有4个与Dog有关的锁。3个是Dog实例的，1个是类的。当你要对静态的方法做同步化时，Java会使用类本身的 锁。因此如果同一个类有两个被同步化过的静态方法，线程需要取得类的锁才能进入这些方法。

问：什么是线程优先级？我听说它可以用来控制调度。

答： 线程的优先级可以对调度器产生影响，但也是没有绝对的保证。优先权的级别会告诉调度器某个线程的重要性，低优先级的线程也许会把机会让给高优先级的线程，也许……建议你可以用优先级来影响执行性能，但绝不能依靠优先级来维持程序的正确性。

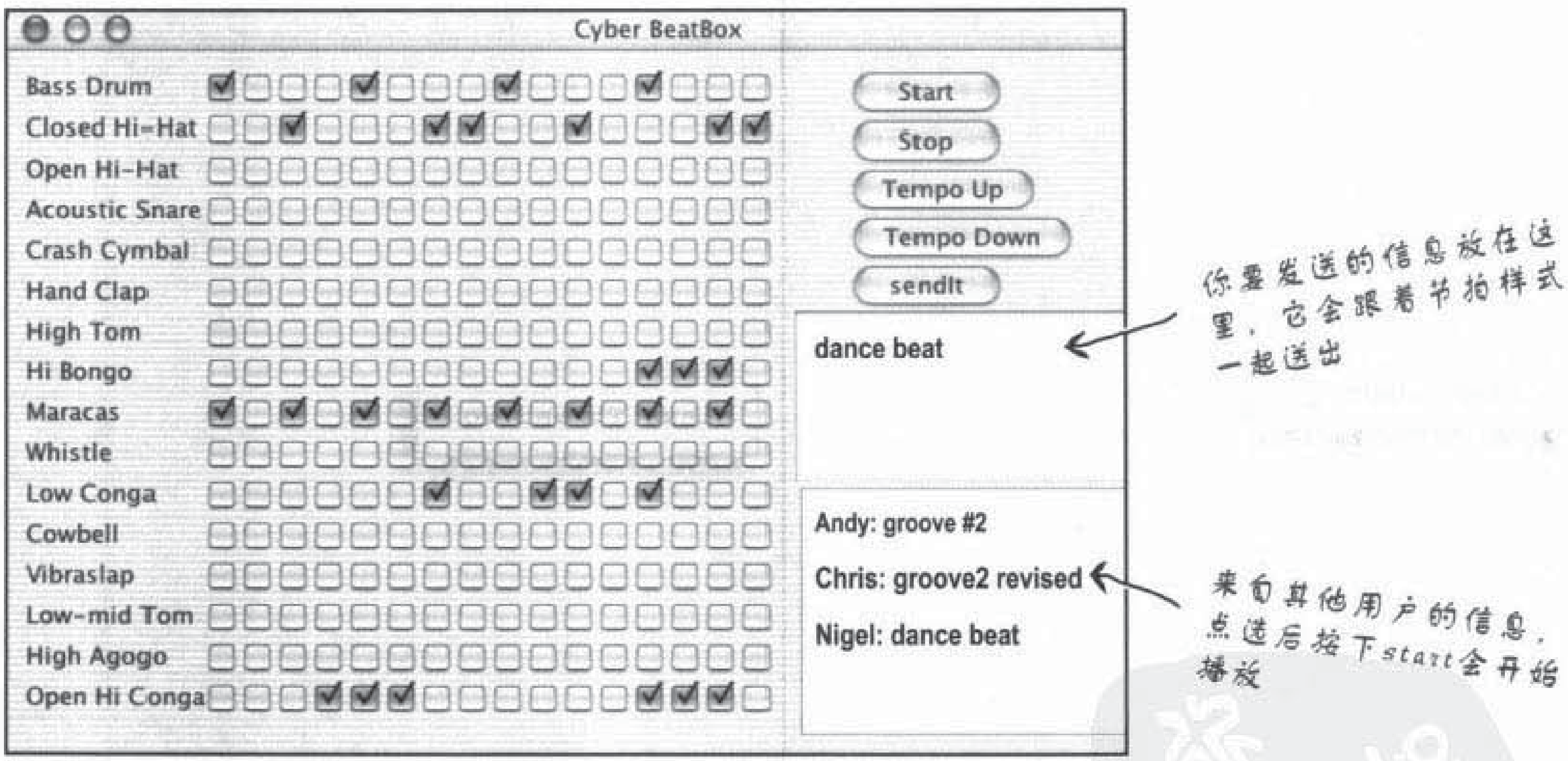
问：为什么不把要被保护的数据的getter与setter都给同步化？例如把账户类上检查余额与提款的两个方法都同步化，而不是把Runnable任务类上的检查余额加上提款操作的这个方法同步化？

答： 事实上，我们应该要把这些方法都同步化，以防止其他的线程以别种方式存取它们。我们没有这么做是因为没有别的程序代码会存取账户余额。

但只是将getter与setter同步化是不够的。要记得同步化的意义是指定某段工作要在不能分割的状态下执行。也就是说单独的操作不重要，重要的是有多个步骤的方法。想想看，如果只是把检查余额单一操作的方法给同步化，杰纶与沛晨的问题同样还是会发生的。因为此时检查余额与提款还是有被分割的可能。

因此把所有存取的方法都同步化是个好主意，但还是得要把不可分割的原子单元作同步化。

程序料理



这是BeatBox的最终决定版！

它能连接到MusicServer上以便发送与接收其他客户端的节拍样式。

程序代码相当长，所以完整的列表是放在附录A。





排排看

下一页有被打散的程序代码。你是否能够重组这些程序代码来产生下面的输出？你可以自己加入括号来保持程序的正确性。

```
public class TestThreads {
```

```
class Accum {
```

```
class ThreadOne
```

```
class ThreadTwo
```

File Edit Window Help Sewing

```
% java TestThreads
one 98098
two 98099
```

排排看（续）

```
Accum a = Accum.getAccum();
Thread one = new Thread(t1);
System.out.println("two "+a.getCount());
} catch(InterruptedException ex) { }
ThreadTwo t2 = new ThreadTwo();
try {
return counter;
counter += add;
Thread two = new Thread(t2);
implements Runnable {
one.start();
Accum a = Accum.getAccum();
Thread.sleep(50);
public static Accum getAccum() {
} catch(InterruptedException ex) { }
private static Accum a = new Accum();
private int counter = 0;
public void run() {
a.updateCounter(1);
Thread.sleep(50);
for(int x=0; x < 99; x++) {
implements Runnable {
a.updateCounter(1000);
public int getCount() {
return a;
System.out.println("one "+a.getCount());
public void updateCounter(int add) {
for(int x=0; x < 98; x++) {
two.start();
public static void main(String [] args) {
try {
public void run() {
private Accum() { }
ThreadOne t1 = new ThreadOne();
```

练习解答

```
public class TestThreads {
  public static void main(String [] args) {
    ThreadOne t1 = new ThreadOne();
    ThreadTwo t2 = new ThreadTwo();
    Thread one = new Thread(t1);
    Thread two = new Thread(t2);
    one.start();
    two.start();
  }
}

class Accum {
  private static Accum a = new Accum();
  private int counter = 0;

  private Accum() { }

  public static Accum getAccum() {
    return a;
  }

  public void updateCounter(int add) {
    counter += add;
  }

  public int getCount() {
    return counter;
  }
}

class ThreadOne implements Runnable {
  Accum a = Accum.getAccum();
  public void run() {
    for(int x=0; x < 98; x++) {
      a.updateCounter(1000);
      try {
        Thread.sleep(50);
      } catch(InterruptedException ex) { }
    }
    System.out.println("one "+a.getCount());
  }
}
```

创建Accum这个类的静态实例

私用的构造函数

两个不同的类会对另一类的同对象作更新，因为两个线程会去存取Accum唯一的实例。

```
private static Accum a = new Accum();
```

上面这行程序会创建Accum的静态实例，而私用的构造函数代表其他人无法创建它的对象。运用这两项技术能够做出称为Singleton的模式，它能限制应用程序上某对象实例的数量（通常会跟名字一样限制一个）。但你也可以使用相同的模式来做出想要的限制。

```
class ThreadTwo implements Runnable {
  Accum a = Accum.getAccum();
  public void run() {
    for(int x=0; x < 99; x++) {
      a.updateCounter(1);
      try {
        Thread.sleep(50);
      } catch(InterruptedException ex) { }
    }
    System.out.println("two "+a.getCount());
  }
}
```



差点出错的气闸舱

温迪正在与开发小组进行设计会议，她看了看窗外的印度洋日出，虽然船上会议室的豪华设备无法掩饰狭小空间带来的封闭感，但破晓时分黑夜与白昼交会刹那的美景还是能缓和她的不安。

5分钟短剧

这一天早上的会议主题是太空站的气闸舱（airlock）控制系统。现在已经接近最后完工的阶段，太空漫游的行程也快要排满了，航天员出入气闸舱的交通将会非常的忙碌。

“早啊，”彼特打了招呼“来得正好，我们马上开始讨论设计细节”。



彼特直接切入主题，一点时间都不浪费。“大家都知道，每个气闸舱内外都有 GUI 操作终端机，航天员可以通过终端机操作气闸程序。”温迪点了点头说道：“彼特，你能不能帮大家说明进入和离开气闸的程序？”彼特转身向白板过去开始画出方法调用顺序。

```
orbiterAirlockExitSequence() //出舱程序

    verifyPortalStatus(); //检查入口状态

    pressurizeAirlock(); //气闸舱加压

    openInnerHatch(); //打开内侧闸门

    confirmAirlockOccupied(); //确认气闸舱有人

    closeInnerHatch(); //关闭内侧闸门

    decompressAirlock(); //气闸舱泄压

    openOuterHatch(); //打开外侧闸门

    confirmAirlockVacated(); //确认气闸舱无人

    closeOuterHatch(); //关闭外侧闸门
```

“为了要确保此过程不会被中断，我们已经把所有被 orbiterAirlockExitSequence( ) 所调用的方法都做了同步化，”彼特边画边说明“我实在很讨厌看到航天员把航天服的裤子脱到一半的样子”。

每个人都同意彼特的说法，但温迪隐隐约约觉得有什么事情不太对劲。“等一下！”温迪知道是什么了“这个程序会出人命的！”

温迪发现了什么？彼特是不是犯了什么错？

温迪发现什么问题？

她发现为了要确保整个离舱程序不会被打断，orbiterAirlockExitSequence()这个方法必须要被同步化。现有的设计会在离舱过程进行到一半的时候被进舱的航天员给打断！个别的操作虽然做同步化，但动作间仍有被插断的可能。整个程序应该要以不可分割的方式进行才能确保外层空间不会有没穿裤子的航天员尸体。

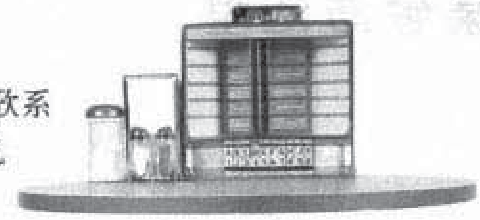
16 集合与泛型

数据结构



排序在Java中只是雕虫小技。 你不必自己编写排序算法就有一大堆现成的工具可供收集与操作数据（除非你正在上电工课，老师铁定会叫你写排序程序的作业）。Java集合框架（Collections Framework）能够支持绝大多数你会用到的数据结构。想要很容易加入元素的列表吗？想要根据名称来搜索吗？打算创建可以自动排除重复项目的列表吗？需要将同事暗算你的次数排个复仇黑名单吗？这里全都有……

记录KTV最常点的歌



欢迎你到大富豪 KTV 担任点歌系统管理员一职。虽然点歌系统没有内置的Java，但是只要有人点歌，数据就会自动记录到一个文本文件中。

你的工作是要管理点播记录、产生报表和管理歌本。你并不需要写出整个程序——很多同事都是找不到程序设计工作才来当服务生的，所以大家可以分工合作，你只需要负责用 Java程序来把数据排序，并且偶尔帮忙打扫包厢就行。因为老板很抠，所以公司没买数据库应用程序，你只能靠内存上的数据集合，还有记录用的文本文件。

你已经知道如何读取与解析文件，并且也用ArrayList来排序。

SongList.txt

```
Communication/The Cardigans
Black Dog/Led Zeppelin
Dreams/Van Halen
Comfortably Numb/Pink Floyd
Beth/Kiss
倒退噜/黄克林
```

这是点歌系统写出的文件，你的程序必须读取文件才能操作数据

挑战一

以歌名来排序

文件上列出歌曲，每行代表一首歌，歌名与歌星用斜线分开，所以应该很容易解析并放到ArrayList上。

消费者通常是看歌名点歌，所以目前只要记歌名就行。

但你会注意到歌曲没有依据字母排序，要怎么办呢？

你知道使用ArrayList的时候，元素会维持被加入ArrayList的顺序，所以它们不会依照字母排序，或许ArrayList这个类有个sort()方法可以用吧？

下面是目前的程序，没有排序功能

```
import java.util.*;
import java.io.*;

public class Jukebox1 {

    ArrayList<String> songList = new ArrayList<String>();

    public static void main(String[] args) {
        new Jukebox1().go();
    }

    public void go() {
        getSongs();
        System.out.println(songList);
    }

    void getSongs() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line =  null;
            while ((line= reader.readLine()) != null) {
                addSong(line);
            }

        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        songList.add(tokens[0]);
    }

}
```

歌曲名称会存在String的ArrayList上

这个方法会载入文件并列出内容

读取文件的程序

split()方法会用反斜线来拆开歌曲内容

因为只需要歌名，所以只取第一项加入SongList

```
File Edit Window Help Dance
%java Jukebox1
[Communication, Black Dog,
Dreams, Comfortably Numb,
Beth, 倒退噜]
```

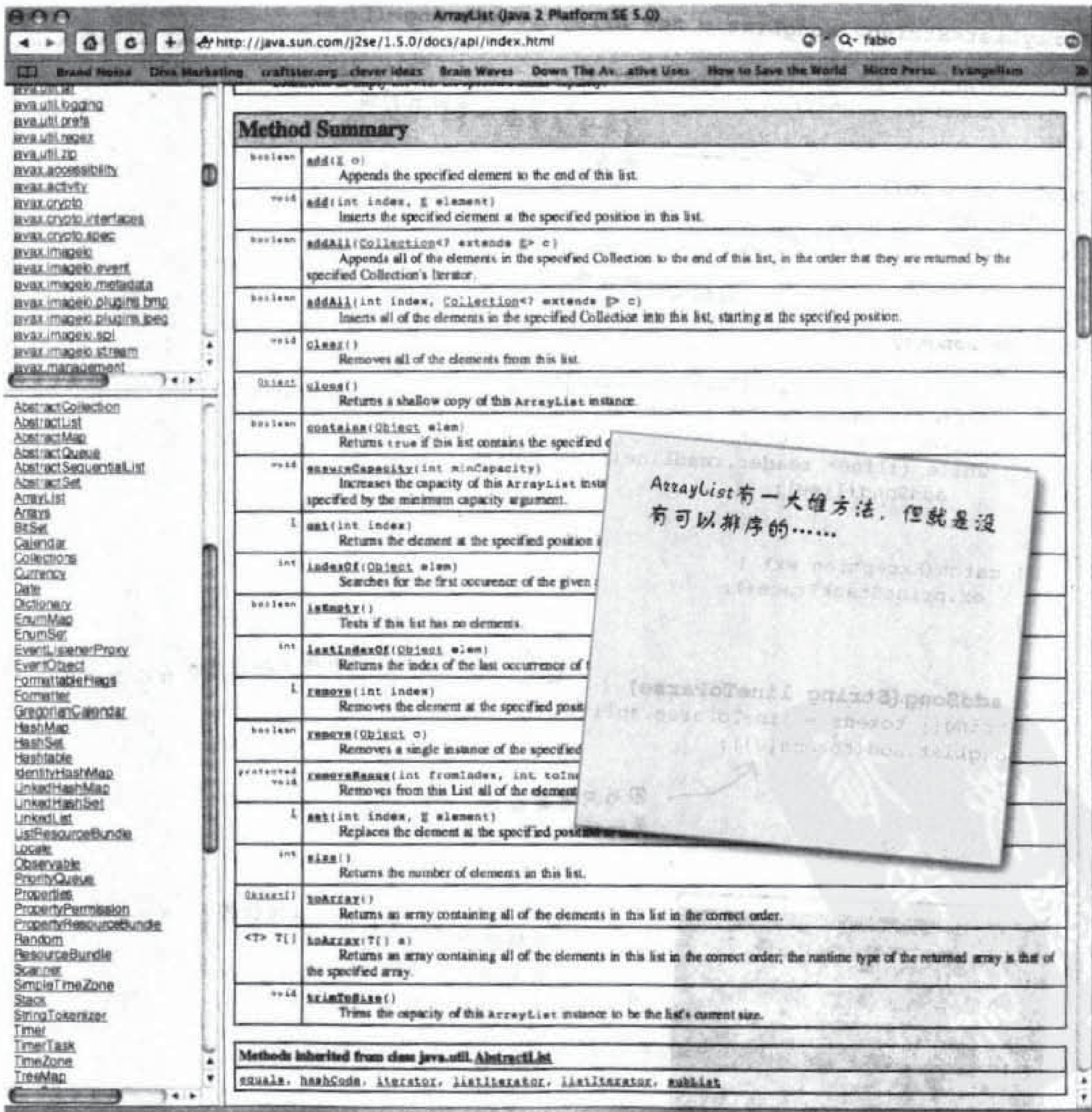
依照加入的顺序列出，与原始的文本文件顺序相同

这肯定不是歌本的排列方式！

但是ArrayList没有sort()这个方法

当你查询ArrayList的说明时，看起来是没有任何方法和排序有关。

查询整个继承结构也没有帮助，很明显ArrayList就是不能排序。





ArrayList不是唯一的集合

虽然ArrayList会是最常用的，但偶尔还是会有特殊情况。下面列出几个较为重要的。

稍后还会有更深入的说明

- **TreeSet**
  以有序状态保持并可防止重复。

- **HashMap**
  可用成对的name/value来保存与取出。

- **LinkedList**
  针对经常插入或删除中间元素所设计的高效率集合。
  （实际上ArrayList还是比较实用）

- **HashSet**
  防止重复的集合，可快速地找寻相符的元素。

- **LinkedHashMap**
  类似HashMap，但可记住元素插入的顺序，也可以设定成依照元素上次存取的先后来排序。

你可以使用TreeSet或Collections.sort()方法

如果你把字符串放进TreeSet而不是ArrayList，这些String会自动地按照字母顺序排在正确的位置。每当你想要列出清单时，元素总是会以字母顺序出现。

当你需要set集合（稍后讨论）或总是会依照字母排列的清单时，它会很好用。

另外一方面，如果你没有需要让清单保持有序的状态，TreeSet的成本会比你想付出的还多——每当插入新项目时，它都必须要花时间找出适当的位置。而ArrayList只要把项目放在最后面就好。

java.util.Collections
public static void copy(List destination, List source)
public static List emptyList()
public static void fill(List listToFill, Object objToFillItWith)
public static int frequency(Collection c, Object o)
public static void reverse(List list)
public static void rotate(List list, int distance)
public static void shuffle(List list)
public static void sort(List list)
public static boole...All(List list, Object oldVal, Object newVal)
// many more me...

嗯……Collections这个类有个sort()方法。它会用到List，而ArrayList有实现List这个接口，所以ArrayList是一个List。感谢多态机制，你确实可以把ArrayList传给用到List的方法。

问： **但你可以用指定的索引来添加新项目到ArrayList中，而不是放到最后面——add()有个重载的版本可以指定int值。这样会比较慢吗？**

答： 是的，插到指定位置会比直接加到最后面要慢。所以add(index, element)不会像add(element)这么快。但通常你不会对ArrayList加上指定的索引。

问： **使用LinkedList这个类会不会比较好？我记得以前上数据结构的课时是这样说的。**

答： 没错，LinkedList对于在中间的插入或删除会比较快，但对大多数的应用程序而言ArrayList与LinkedList的差异有限，除非元素量真的很大。稍后会讨论LinkedList。

注意：这不是完整的API说明，我们把稍后会讨论的generic信息拿掉来加以简化。

对点歌系统加上 Collections.sort()

Collections.sort()会把list中的String依照字母排序

```
import java.util.*;
import java.io.*;

public class Jukebox1 {

    ArrayList<String> songList = new ArrayList<String>();

    public static void main(String[] args) {
        new Jukebox1().go();
    }

    public void go() {
        getSongs();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }

    void getSongs() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line =  null;
            while ((line= reader.readLine()) != null) {
                addSong(line);
            }

        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        songList.add(tokens[0]);
    }
}
```

调用Collection静态的sort()然后再列出清单。第二次的输出会依照字母排序！

```
File Edit Window Help Chill
%java Jukebox1

[Communication, Black Dog, Dreams, Comfortably Numb,
Beth, 倒退嚕]

[Beth, Black Dog, Comfortably Numb,Communication,
Dreams, 倒退嚕]
```

调用前

调用后

但是现在要用Song对象而不只是String

老板说list里面要摆的是Song这个类的实例，这样新的点歌系统才会有更细节的资料可以输出。所以文件内也会从两种数据增加到4种。

Song这个类是很单纯的，但有一项很有意思的功能：被覆盖过的toString()。要知道toString()是定义在Object这个类中，所以Java中的每个类都有继承到，且因为对象被System.out.println(anObject))列出来时会被调用toString()，所以当你要把list列出时，每个对象的toString()都会被调用一次。

SongListMore.txt

```
Communication/The
Cardigans/5/80
Black Dog/Led Zeppelin/4/84
Dreams/Van Halen/6/120
Comfortably Numb/Pink
Floyd/5/110
Beth/Kiss/4/100
倒退噜/黄克林/5/90
```

新的歌曲文件带有4项属性，所以我们需要创建出Song的实例变量来带这些属性。

```
class Song {
    String title;
    String artist;
    String rating;
    String bpm;

    Song(String t, String a, String r, String b) {
        title = t;
        artist = a;
        rating = r;
        bpm = b;
    }

    public String getTitle() {
        return title;
    }

    public String getArtist() {
        return artist;
    }

    public String getRating() {
        return rating;
    }

    public String getBpm() {
        return bpm;
    }

    public String toString() {
        return title;
    }
}
```

对应4种属性的4个实例变量

变量都会在创建时从构造函数中设定

4种属性的getter

将toString()覆盖过，让它返回歌名

修改点歌系统程序

程序代码只有改一点点，文件输入/输出的部分不变，除了属性变成4个之外，解析的部分也一样使用String.split()。当然ArrayList的类型要从<String>改成<Song>。

将String改成Song类型

```
import java.util.*;
import java.io.*;

public class Jukebox3 {

    ArrayList<Song> songList = new ArrayList<Song>();
    public static void main(String[] args) {
        new Jukebox3().go();
    }
    public void go() {
        getSongs();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }
    void getSongs() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line = null;
            while ((line= reader.readLine()) != null) {
                addSong(line);
            }
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }

    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");

        Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
        songList.add(nextSong);
    }
}
```

使用解析出来的4项属性来创建Song对象并加入到list中

无法通过编译！

有点问题，Collections类很明显地说明sort( )这个方法会取用List。

ArrayList是一个List，因为ArrayList有实现List这个接口，所以应该要没问题才对。

但就是不行！

编译器表示它找不到取用ArrayList<Song>参数的sort( )方法，所以ArrayList<String>与ArrayList<Song>之间到底有什么差异？为什么编译器不会让它过关？

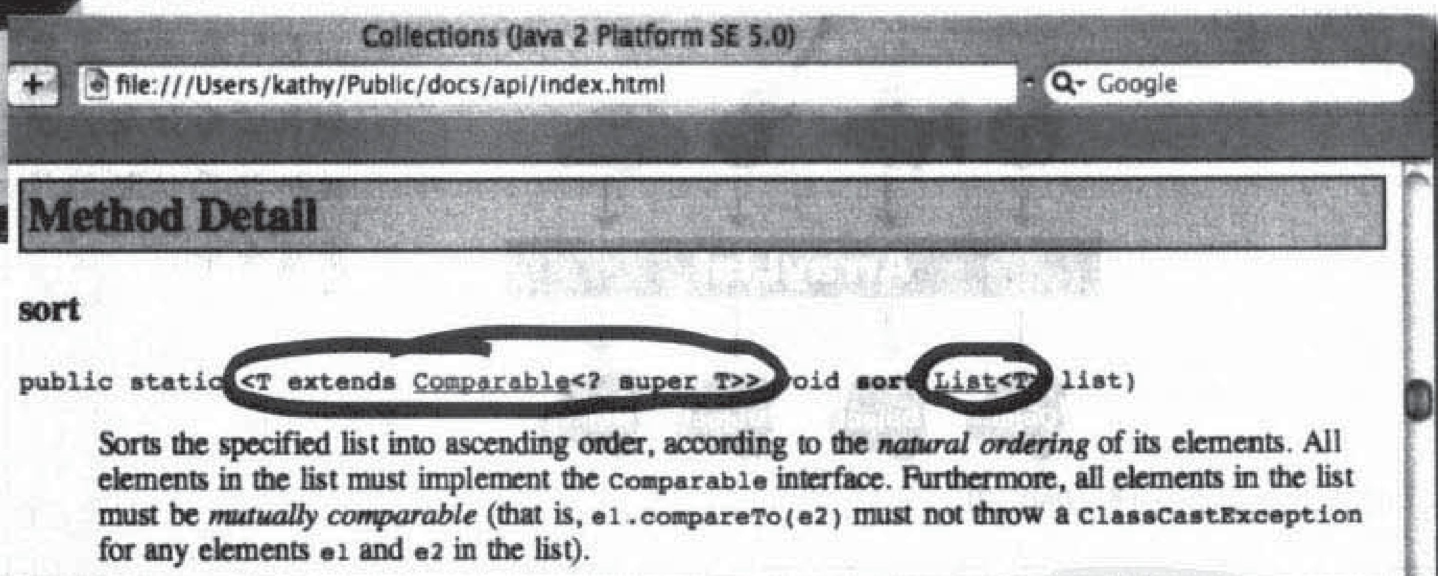
```
File Edit Window Help Bummer
%javac Jukebox3.java
Jukebox3.java:15: cannot find symbol
symbol  : method sort(java.util.ArrayList<Song>)
location: class java.util.Collections
                Collections.sort(songList);
                           ^
1 error
```

或许你已经在想："那它要靠什么东西来排序？" sort()要怎么判断某首歌应该在另外一首歌之前？很明显的，如果你想要让歌曲依照曲名字母排列，就得要有一种方法可以告诉sort()它依靠的不是曲名长度来排列。

接下来我们会有几页进一步的讨论，但首先要解决无法通过编译器的问题。



sort()的声明


Collections (Java 2 Platform SE 5.0)

file:///Users/kathy/Public/docs/api/index.html

Google

Method Detail

sort

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the *natural ordering* of its elements. All elements in the list must implement the `Comparable` interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

从API说明文件找java.util.Collections下面的sort()，你会发现它的声明有点怪怪的。至少跟我们之前所看过的有些不一样。

这是因为sort()很大量地运用到泛型（generic）功能。只要你在Java的程序或文件中看到<>这一组符号，就代表泛型正在作用——它是一种从Java 5.0开始加入的特质。看起来我们得先学会如何解读说明文件才能看得出来为何ArrayList可应付String对象，但不吃Song对象这一套。

泛型意味着更好的类型安全性

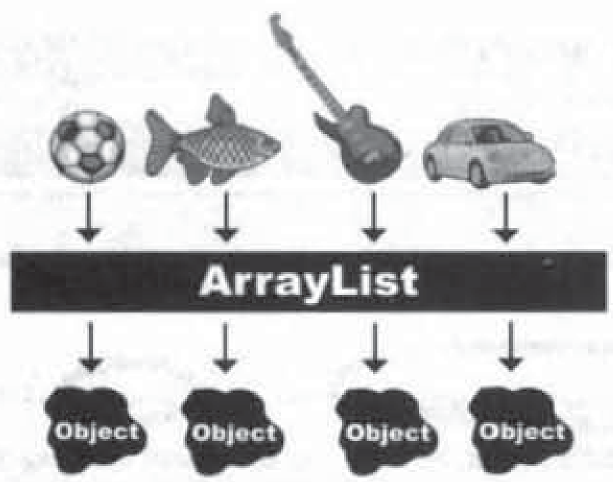
我们就这么说吧，几乎所有你会以泛型写的程序都与处理集合有关。虽然泛型可以用在其他地方，但它主要目的还是让你能够写出有类型安全性的集合。也就是说，让编译器能够帮忙防止你把Dog加到一群Cat中。

在泛型功能出现前，编译器无法注意到你加入集合中的东西是什么，因为所有的集合都写成处理Object类型。你可以把任何东西放进ArrayList中，有点像是ArrayList<Object>。

没有泛型

各种对象以引用的形式加入到ArrayList中。

在泛型出现前，没有办法声明ArrayList的类型，所以只能用Object来操作



出来时会是Object类型的引用。

运用泛型你就可以创建类型安全更好的集合，让问题尽可能在编译期就能抓到，而不会等到执行期才冒出来。

如果没有泛型，编译器会很愉快地接受你把绵羊对象送到老虎集合中。

使用泛型

仅有Fish对象能以引用形式加入到ArrayList中。



加上泛型之后，你不能把非Fish放进ArrayList<Fish>中，因此取出的也会是Fish引用。

出来的还是Fish对象的引用。

关于泛型

泛型有好几样东西需要知道，但对大部分的程序员来说，其实只有3件事情是重要的：

1. **创建被泛型化类（例如ArrayList）的实例。**

   创建ArrayList时你必须要指定它所容许的对象，就像单纯的数组那样。

```
new ArrayList<Song>()
```

2. **声明与指定泛型类型的变量。**

   多态遇到泛型类型会怎样？如果你有个ArrayList<Animal>引用变量，能够赋给ArrayList<Dog>吗？如果是List<Animal>呢？可以赋给ArrayList<Animal>吗？稍后分晓……

```
List<Song> songList =
      new ArrayList<Song>()
```

3. **声明（与调用）取用泛型类型的方法。**

   如果你有个方法取用Animal对象ArrayList的参数，那代表什么？是否也可以传入Dog对象的ArrayList呢？接着我们会讨论到某些多态问题的细节内容。

   （其实这跟第二项一样，只是让你知道我们认为这有多重要。）

```
void foo(List<Song> list)

x.foo(songList)
```

问： **但我不是还需要学习如何自己创建泛型的类吗？如果我想设计出让人们在初始化同时要决定类型的类要怎么办？**

答： 你或许不会经常做这件事。想想看，API 的设计团队已经涵盖了大部分你会遇到的数据结构，而几乎只有集合才会真的需要泛型。也就是说这些类是设计来保存其他元素，并要让程序员在声明与初始化类的时候指定元素的类型。

没错，你可能会想要创建出泛型的类，但那是很少见的情况，所以我们就不多做讨论了（还是可以从这些内容看出大概）。

使用泛型的类

因为ArrayList是最常用的泛型化类型，我们会从查看它的文件看起。有两个关键的部分：

（1）类的声明。

（2）新增元素的方法的声明。

把E想做是“集合所要维护和返回的元素类型”

（E代表Element）

ArrayList的说明文件（又称“E是什么？”）

E部分会用你所声明与创建的真正类型来取代

ArrayList是AbstractList的子类，所以指定给ArrayList的类型会自动地用在AbstractList上

```
public class ArrayList<E> extends AbstractList<E> implements List<E>  ... {

    public boolean add(E o)

    // 更多代码

}
```

这边最重要！E用来指示可以加入ArrayList的元素类型

此类型也会用在List这个接口上

E代表用来创建与初始ArrayList的类型。当你看到ArrayList文件上的E时，就可以把它换成实际上的类型。

所以ArrayList<Song>就会在所有方法与变量的声明中把E换成Song。

ArrayList的类型参数

下面这行程序：

```
ArrayList<String> thisList = new ArrayList<String>
```

代表这个ArrayList：

```
public class ArrayList<E> extends AbstractList<E> ... {

      public boolean add(E o)
      // 更多代码
}
```

会被编译器这样看待：

```
public class ArrayList<String> extends AbstractList<String>... {

      public boolean add(String o)
      // 更多代码
}
```

也就是说，E会被所指定的真正类型所取代（又被称为类型参数）。这也是为何add()这个方法不会让你加入与E所指定类型不兼容的引用的原因。若你创建出ArrayList<String>，则add()会变成add(String o)。若你创建出ArrayList<Dog>，则add会变成add(Dog o)。

问： 只能用E吗？因为排序的文件上面用的是T……

答： 你可以使用任何合法的Java标识字符串。这代表不管用什么都会被当作是类型参数。但习惯用法是以单一的字母表示（你也应该这么做），除非与集合有关，否则都是用T，因为E很清楚地指明是元素。

运用泛型的方法

泛型的类代表类的声明用到类型参数。泛型的方法代表方法的声明特征用到类型参数。

在方法中的类型参数有几种不同的运用方式。

1. 使用定义在类声明的类型参数。

```
public class ArrayList<E> extends AbstractList<E> ... {
    public boolean add(E o)
```

只能在此使用E，因为它已经被定义成类的一部分

当你声明类的类型参数时，你就可以直接把该类或接口类型用在任何地方。参数的类型声明基本上会以用来初始化类的类型来取代。

2. 使用未定义在类声明的类型参数。

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

因为在前面声明T所以这里就可以使用<T>

如果类本身没有使用类型参数，你还是可以通过在一个不寻常但可行的位置上指定给方法——在返回类型之前。这个方法意味着T可以是“任何一种 Animal”。

PDG



这就是怪异之处……

这行程序：

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

跟这个是不一样的：

```
public void takeThing(ArrayList<Animal> list)
```

两者都合法，但意义不同！

首先，**<T extends Animal>**是方法声明的一部分，表示任何被声明为Animal或Animal的子型（像是Cat或Dog）的ArrayList是合法的。因此你可以使用ArrayList<Dog>、ArrayList<Cat>或ArrayList<Animal>来调用上面的方法。

但是，下面方法的参数是ArrayList<Animal> list，代表只有ArrayList<Animal>是合法的。也就是说第一个可以使用任何一种Animal 的ArrayList，而第二个方法只能使用 Animal 的 ArrayList。

没错，这看起来已经违反动态绑定的精神，但在这一章最后的回顾时就会很清楚这是怎么一回事。现在只要记得我们还在想办法对 SongList 排序就行。

现在只要知道上面的语法是合法的就够了，它代表你可以传入以 Animal 或子型来初始化的 ArrayList 对象。

接着回头看sort( )方法……



出错的地方……

```
File Edit Window Help Bummer
%javac Jukebox3.java
Jukebox3.java:15: cannot find symbol
symbol   : method sort(java.util.ArrayList<Song>)
location: class java.util.Collections
                    Collections.sort(songList);
                               ^
1 error
```

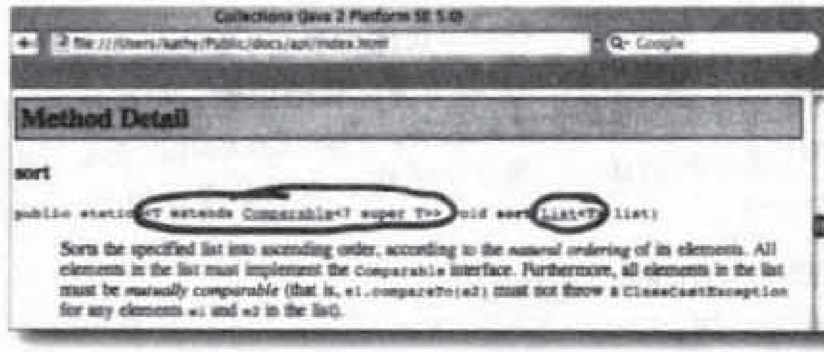
```
import java.util.*;
import java.io.*;

public class Jukebox3 {
    ArrayList<Song> songList = new ArrayList<Song>();
    public static void main(String[] args) {
        new Jukebox3().go();
    }
    public void go() {
        getSongs();
        System.out.println(songList);
        Collections.sort(songList);
        System.out.println(songList);
    }
    void getSongs() {
        try {
            File file = new File("SongList.txt");
            BufferedReader reader = new BufferedReader(new FileReader(file));
            String line =  null;
            while ((line= reader.readLine()) != null) {
                addSong(line);
            }
        } catch(Exception ex) {
            ex.printStackTrace();
        }
    }
    void addSong(String lineToParse) {
        String[] tokens = lineToParse.split("/");
        Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
        songList.add(nextSong);
    }
}
```

就是这里有问题！传入ArrayList<String>可以过关，但ArrayList<Song>就不行

再看一下sort()方法

查找文件上关于为何对String的list排序可行，但Song对象就不行的线索。看起来答案应该是：



看起来sort()方法只能接受Comparable对象的list。

Song不是Comparable的子型，所以你无法对Song的list排序。

至少还不行……

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

这表示它必须是Comparable

先不管它，它代表Comparable的类型参数必须是T或T的父型

仅能传入继承Comparable的参数化类型的list

呃……我刚查过String的说明，它没有继承过Comparable，只有实现，因为Comparable是个接口，所以上面的extends不太合理。



```
public final class String extends Object implements Serializable,
                                          Comparable<String>, CharSequence
```

以泛型的观点来说，extend代表extend或implement

Java设计团队有给你一种对参数化类型加上限制的方法，因此你可以加上只能使用Animal的子类之类的限制。但你也会需要限制只允许有实现某特定接口的类。因此现在的状况需要对两种情形都能适用的语法——继承和实现。也就是说适用于extends和implements。

答案揭晓：extends。它确实代表“是一个……”，且不管是接口或类都能适用。

对泛型来说，extends这个关键词代表“是一个……”，且适用于类和接口。

Comparable是个接口，所以这可以读作：“T必须是有实现Comparable的类型”

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

这是类或接口都没关系，还是可以说是extends过的

问： 为什么不弄个“is”关键词来用？

答： 对语言加入新的关键词是很重大是事件，因为这样可能会有破坏较早写作程序的风险。想想看，假如你有用过名称为is的变量（我们就有用is来代表输入流）会怎样。且因为关键词是不能用来当作标识符，所以本来可以过关的程序也会因为用了保留字而被退回。因此只要有可能的话，Sun的工程师就会像extends这样反复利用现有的关键词。但有时候连他们也没选择的余地……

有少数几个关键词被加入语言中，像是Java 1.4出现的**assert**和Java 5.0的**enum**（见附录）。这确实会破坏某些程序，但有时还是可以对新版 Java加上选项让它仿真旧版的行为。通过指定特殊标记给编译器或Java虚拟机就可以让它们如此执行。

（在命令栏上输入javac或java且不加任何指令就会看到可用的选项，讨论部署的章节会有更多的细节。）

知道哪里有问题了……
Song类必须实现Comparable

我们只有在Song类实现Comparable的情况下才能把ArrayList<Song>传给sort()方法，因为这个方法就是如此声明的。稍微看过一下说明文件就会知道Comparable其实很单纯，只有一个方法需要实现。

java.lang.Comparable

```
public interface Comparable<T> {
   int compareTo(T o);
}
```

而compareTo()的文件说明是这样的：

Returns:

```
a negative integer, zero, or a
positive integer as this object
is less than, equal to, or greater
than the specified object.
```

看起来compareTo()方法是会从某个Song的对象来调用，然后传入其他Song对象的引用。执行compareTo()方法的Song必须要判别在排序位置上它自己是高于、低于或相等于所传入的Song。

你的主要任务就是决定如何判断Song的先后，然后以compareTo()方法的实现来反映出这个逻辑。返回负数值表示传入的Song大于执行的Song、正数刚好相反，而返回0代表相等（以排序的目的来说，并不代表两对象真的相等）。

重点在于：两个Song是如何比较出大小的？

要先确定如何比较才能有办法实现Comparable这个接口。

Sharpen your pencil

写出实现compareTo()方法的程序代码以让Song对象能够依据歌名来排序。

提示：如果你没有搞错的话，大约在3行之内就可以搞定。

更新、更好、更comparable的Song类

我们决定要靠歌名来排序，所以把compareTo()方法实现成用执行方法的Song 歌曲名称和所传入的Song歌曲名称来比较。也就是说，执行方法的Song 会判断它的歌名和参数歌名的比较结果。

我们知道String一定有办法比较字母先后顺序，因为sort()方法就可以比较String的list。我们也知道String有个compareTo()方法，因此只要让曲名String相互比较就好，不必另行编写比较字母的算法！

通常箭头所指处会是一样的……为了排序的目的，大象只会跟大象比较，而不是跟鸡腿比大小

要比较的对象

就是这么简单！返回String比较的结果就行

```
class Song implements Comparable<Song> {
   String title;
   String artist;
   String rating;
   String bpm;

   public int compareTo(Song s) {
       return title.compareTo(s.getTitle());
   }

   Song(String t, String a, String r, String b) {
      title = t;
      artist = a;
      rating = r;
      bpm = b;
   }

   public String getTitle() {
       return title;
   }

   public String getArtist() {
      return artist;
   }

   public String getRating() {
      return rating;
   }

  public String getBpm() {
    return bpm;
  }

  public String toString() {
     return title;
  }
}
```

这次成功了，调用sort()方法之后会把Song依照字母作排序

```
File Edit Window Help Ambient
%java Jukebox3
[Communication, Black Dog, Dreams, Comfortably Numb,
Beth, 倒退噜]
[Beth, Black Dog, Comfortably Numb,Communication,
Dreams, 倒退噜]
```

list排好了，但是……



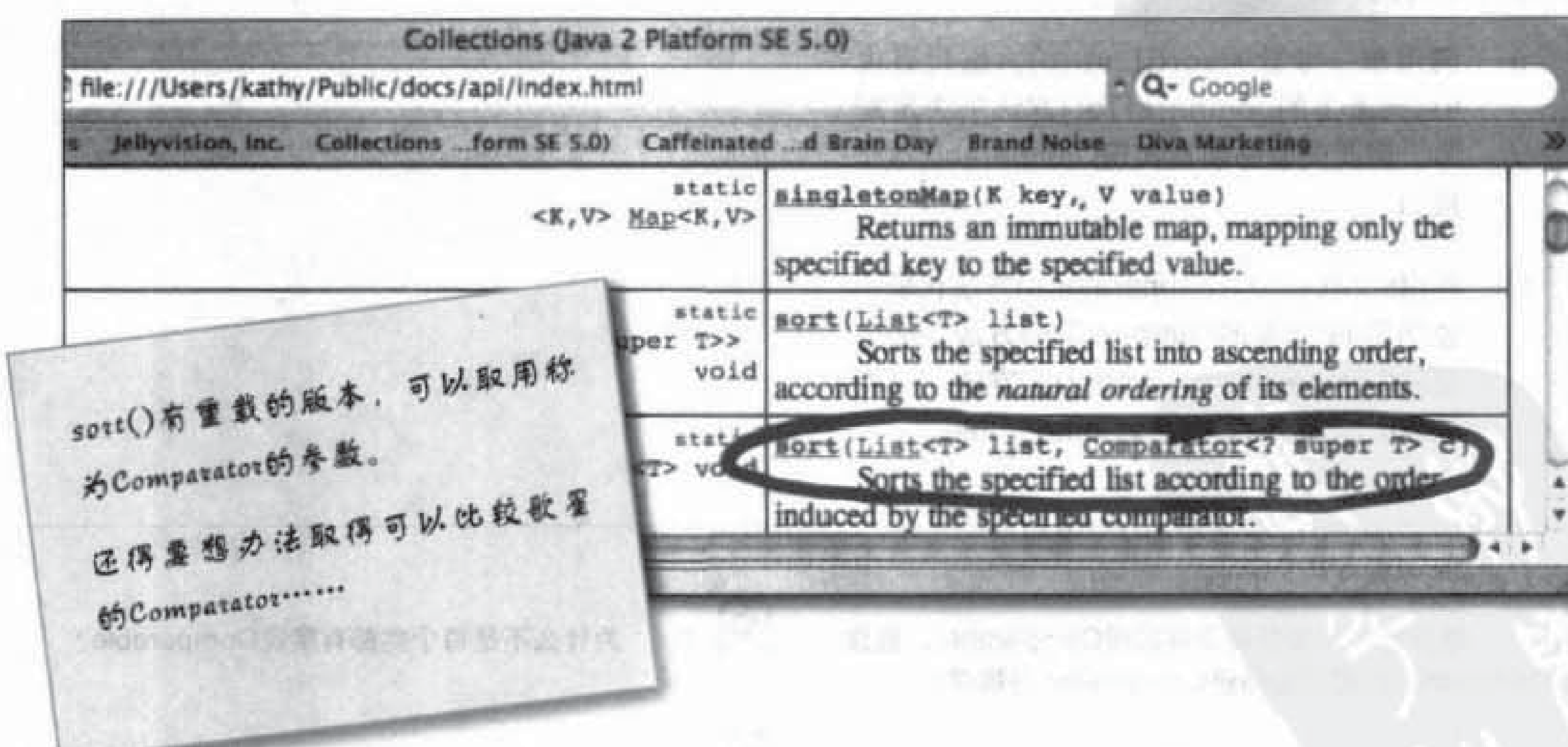
又有问题了，大厨说他只会唱陈雷的歌，所以除了依照歌名排序之外，也要能依照歌星名来排。

但是集合元素的排序就只能实现一个comparetTo()，那要怎么办？

有一种糟糕的做法是在Song的类中加上一个旗标，然后在comparetTo( )中再加上if判断来依照标识决定用哪个项目做比较。

这样很不好，厨房管到外场，简直是造反……事实上还有更好的方法。API中已经设计出一种方法来解决这种问题——以不同方式来比大小。

继续查询API，有另一种sort()方法——它取用Comparator参数。



使用自制的Comparator

使用compareTo()方法时，list中的元素只能有一种将自己与同类型的另一个元素做比较的方法。但Comparator是独立于所比较元素类型之外的——它是独立的类。因此你可以有各种不同的比较方法！想要依照歌星排序吗？做个ArtistComparator，想要依照恶搞程度排序吗？没问题，做个KusoComparator。

然后只要调用重载版，取用List与Comparator参数的sort()方法来处理就可以。

取用Comparator版的sort()方法会用Comparator而不是元素内置的compareTo()方法来比较顺序。也就是说，如果sort()方法带有Comparator，它就不会调用元素的compareTo()方法，而会去调用Comparator的compare()方法。

所以规则是这样的：

- **调用单一参数的sort(List o)方法代表由list元素上的comparetTo()方法来决定顺序。因此元素必须要实现Comparable这个接口。**

- **调用sort(List o, Comparator c)方法代表不会调用list元素的compareTo()方法，而会使用Comparator的compare()方法。这意味着list元素不需要实现Comparable。**

java.util.Comparator

```
public interface Comparator<T> {
   int compare(T o1, T o2);
}
```

如果传Comparator给sort()方法，则排序是由Comparator而不是元素的compareTo()方法来决定。

问： 是否这代表如果类没有实现Comparable，且你也没拿到原始码，还是能够通过Comparator 来排序？

答： 没错。另外一种方法就是子类化该元素并实现出 Comparable。

问： 为什么不是每个类都有实现Comparable？

答： 你真的认为每种东西都可以排序吗？如果不能排序的东西硬加上 Comparable只会使人产生误解。

用Comparator更新点歌系统

我们在新版本做了3件事：

(1) 创建并实现Comparator的内部类，以compare()方法取代compareTo()方法。

(2) 制作该类的实例。

(3) 调用重载版的sort()，传入歌曲的list以及Comparator的实例。

附注：我们也有更新Song的toString()以列出歌名与歌星。

```
import java.util.*;
import java.io.*;

public class Jukebox5 {
   ArrayList<Song> songList = new ArrayList<Song>();
   public static void main(String[] args) {
      new Jukebox5().go();
   }

   class ArtistCompare implements Comparator<Song> {
     public int compare(Song one, Song two) {
         return one.getArtist().compareTo(two.getArtist());
     }
   }

   public void go() {
     getSongs();
     System.out.println(songList);
     Collections.sort(songList);
     System.out.println(songList);

     ArtistCompare  artistCompare = new ArtistCompare();
     Collections.sort(songList, artistCompare);

     System.out.println(songList);
   }

   void getSongs() {
       // I/O程序代码
   }

   void addSong(String lineToParse) {
       // 解析歌曲清单
   }
}
```

创建并实现Comparator的内部类，注意到类型参数和要比较的类型是相符的

这会返回String

以String来比较

创建Comparator的实例

调用sort()，传入list与Comparator对象

（译注：这就是所谓的垃圾注释，好注释不需重复程序代码就已经明白表达出的东西。）

> 注意：另外一种设计哲学是拿掉Song的compareTo()，以两个Comparator来实现歌名排序和歌星名排序。这代表我们只能使用两个参数版的Collections.sort( )。

```
import ___________________;

public class SortMountains {

  LinkedList______________ mtn = new LinkedList____________();

  class NameCompare _________________________________ {
    public int compare(Mountain one, Mountain two) {
      return _________________________;
    }
  }
  class HeightCompare _____________________________ {
    public int compare(Mountain one, Mountain two) {
      return (________________________);
    }
  }
  public static void main(String [] args) {
    new SortMountains().go();
  }
  public void go() {
    mtn.add(new Mountain("Longs", 14255));
    mtn.add(new Mountain("Elbert", 14433));
    mtn.add(new Mountain("Maroon", 14156));
    mtn.add(new Mountain("Castle", 14265));
    System.out.println("as entered:\n" + mtn);
    NameCompare nc = new NameCompare();

    _________________________________;
    System.out.println("by name:\n" + mtn);
    HeightCompare hc = new HeightCompare();

    _________________________________;
    System.out.println("by height:\n" + mtn);
  }
}

class Mountain {

  __________________;

  _________________;

  ______________________ {

    _________________;

    _________________;

  }
  _________________________ {

    _____________________________;

  }
}
```

Sharpen your pencil

逆向工程

假设程序代码都在同一个文件。你的任务是要填满空格让程序产生下面的输出。

答案在本章最后面。

输出：

```
File Edit Window Help ThisOne'sForBob
%java SortMountains
as entered:
[Longs 14255, Elbert 14433, Maroon 14156, Castle 14265]
by name:
[Castle 14265, Elbert 14433, Longs 14255, Maroon 14156]
by height:
[Elbert 14433, Castle 14265, Longs 14255, Maroon 14156]
```

Sharpen your pencil

填空题

回答下面的问题，答案只能从可用的答案中挑出。标准答案在章末。

可用的答案：

Comparator,
Comparable,
compareTo( ),
compare( ),
yes,
no

对下面这行程序来说：

```
Collections.sort(myArrayList);
```

（1）存储在myArrayList中对象的class一定要实现什么？ ______________

（2）存储在myArrayList中对象的class一定要实现什么方法？ ______________

（3）存储在myArrayList中对象的 class 能够同时实现Comparator和Comparable吗？ ________

对下面这行程序来说：

```
Collection.sort(myArrayList, myCompare);
```

（4）存储在myArrayList中对象的class必须实现 Comparable吗？ ______________

（5）存储在myArrayList中对象的class必须实现Comparator 吗？ ______________

（6）存储在myArrayList中对象的class必须实现Comparable 吗？ ______________

（7）存储在myArrayList中对象的class必须实现Comparator 吗？ ______________

（8）myCompare必须实现什么？ ______________

（9）myCompare必须实现什么方法？ ______________

PDG

老爷不好了！数据有重复……

排序功能工作得很好，现在我们知道要如何对歌名和歌星名来排序。但是在测试点歌系统的文本文件时发现到新的问题——排序过的列表带有重复的数据。

这可能是因为点歌系统持续写入文件而不管同样的歌曲是否早就被写入到文本文件中。SongListMore.txt 是完整的点歌记录，所以带有重复的歌曲记录是很正常的，就好像汽车维修员身上带有扳手也是很自然的一样。

```
File Edit Window Help TooManyNotes
%java Jukebox4

[Pink Moon: Nick Drake, Somersault: Zero 7, Shiva Moon: Prem
Joshua, Circles: BT, Deep Channel: Afro Celts, Passenger: Headmix,
Listen: Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Circles:
BT]

[Circles: BT, Circles: BT, Deep Channel: Afro Celts, Listen:
Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Passenger:
Headmix, Pink Moon: Nick Drake, Shiva Moon: Prem Joshua,
Somersault: Zero 7]

[Deep Channel: Afro Celts, Circles: BT, Circles: BT, Passenger:
Headmix, Pink Moon: Nick Drake, Shiva Moon: Prem Joshua, Listen:
Tahiti 80, Listen: Tahiti 80, Listen: Tahiti 80, Somersault: Zero
7]
```

排序前

依照歌名排序

依照歌星名排序

SongListMore.txt

```
Pink Moon/Nick Drake/5/80
Somersault/Zero 7/4/84
Shiva Moon/Prem Joshua/6/120
Circles/BT/5/110
Deep Channel/Afro Celts/4/120
Passenger/Headmix/4/100
Listen/Tahiti 80/5/90
Listen/Tahiti 80/5/90
Listen/Tahiti 80/5/90
Circles/BT/5/110
```

这个文件带有重复的歌曲，因为点歌系统会依序地写入记录。有人不停地重复点同一首歌来听，所以就会连续重复的记录。

因为要保持记录完整，所以这个部分不能修改，只好改我们自己的程序代码。

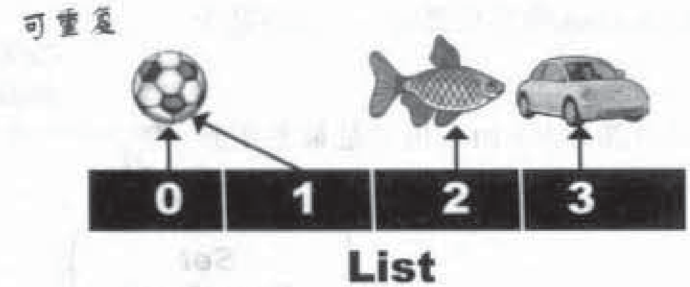
我们需要Set集合

从 Collection 的 API 说明文件中我们发现3个主要的接口：List、Set和Map。
ArrayList 是个 List，但看起来 Set 才是我们所需要的。

➤ **LIST**：对付顺序的好帮手。

是一种知道索引位置的集合。

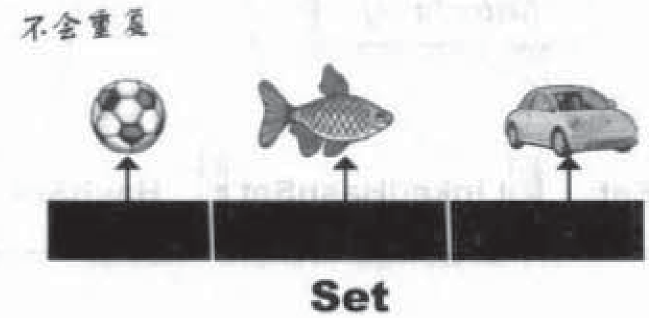
List 知道某物在系列集合中的位置。可以有多个元素引用相同的对象。



➤ **SET**：注重独一无二的性质。

不允许重复的集合。

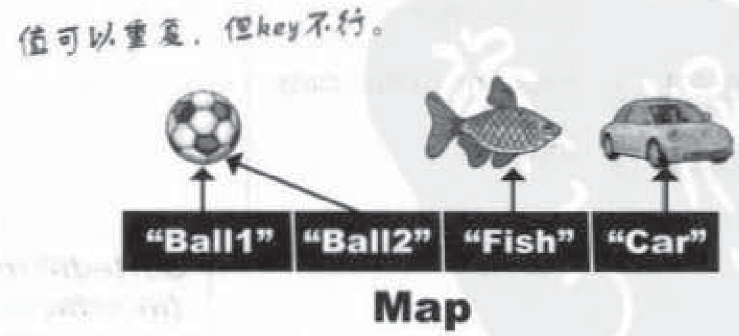
它知道某物是否已经存在于集合中。不会有多个元素引用相同的对象（被认为相等的两个对象也不行，稍后有更多的说明）。



➤ **MAP**：用 key 来搜索的专家。

使用成对的键值和数据值。

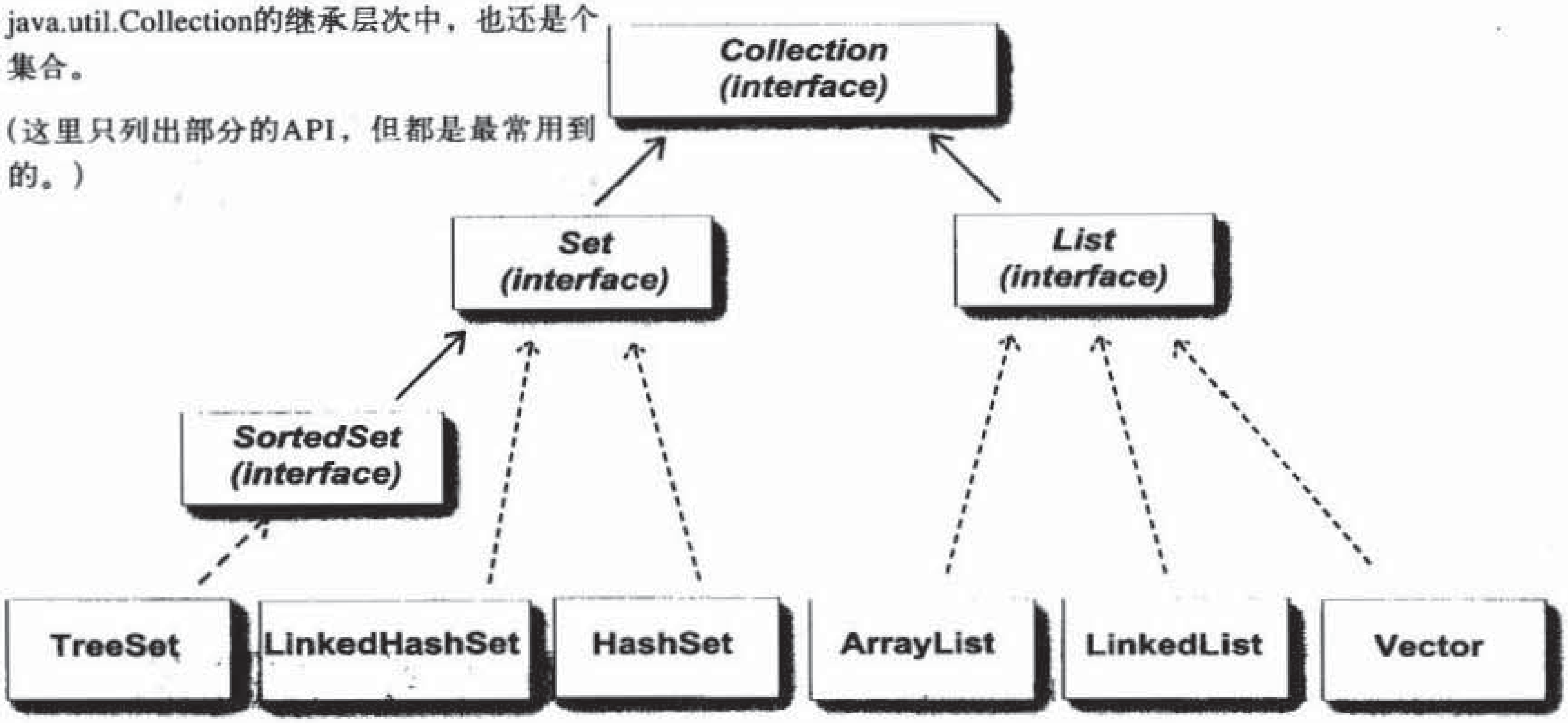
Map 会维护与 key 有关联的值。两个 key 可以引用相同的对象，但 key 不能重复，典型的 key 会是 String，但也可以是任何对象。

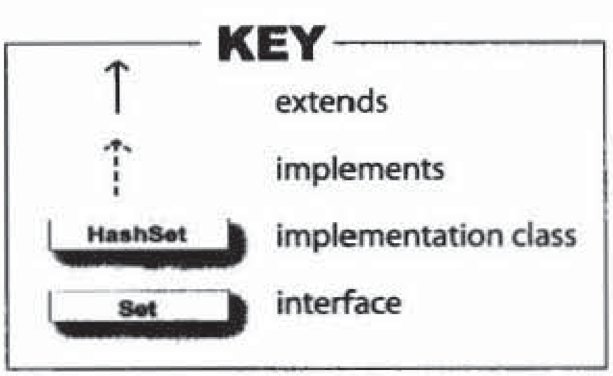


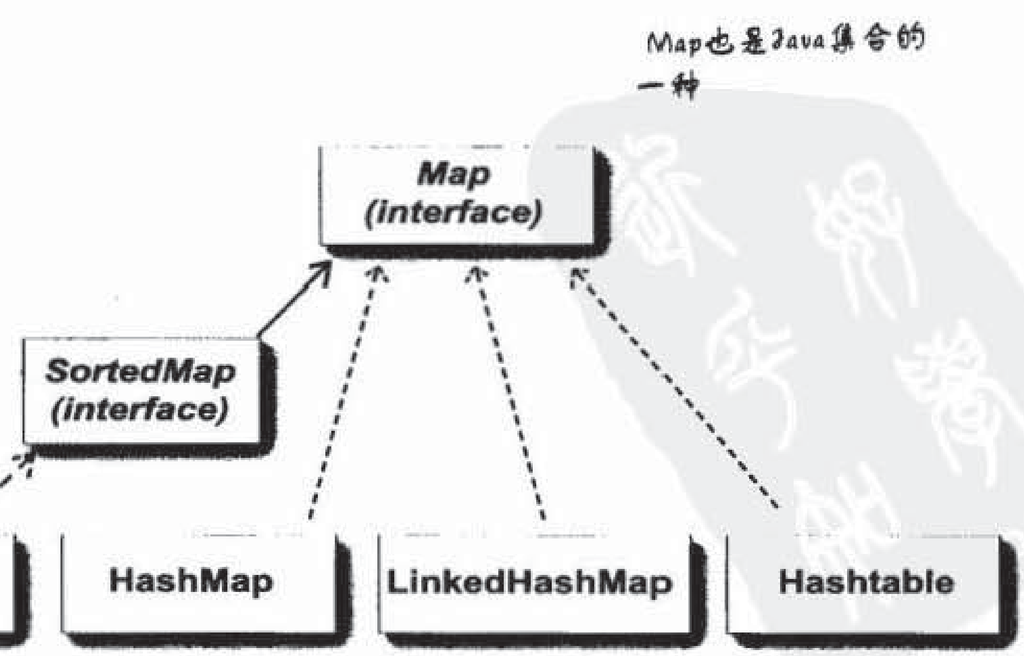
Collection API（摘列）

注意Map事实上并没有继承Collection这个接口，但Map也应该被当作是Collection Framework中的一分子。因此就算它不在java.util.Collection的继承层次中，也还是个集合。

（这里只列出部分的API，但都是最常用到的。）







以 HashSet 取代 ArrayList

我们把点歌系统改成使用HashSet。这里列出一部分的程序代码，其余的部分可以沿用上一版。为了易于阅读，输出的部分直接使用toString( )。

```
import java.util.*;
import java.io.*;

public class Jukebox6 {
   ArrayList<Song> songList = new ArrayList<Song>();
   // main method etc.

   public void go() {
      getSongs();
      System.out.println(songList);
      Collections.sort(songList);
      System.out.println(songList);

      HashSet<Song> songSet = new HashSet<Song>();
      songSet.addAll(songList);
      System.out.println(songSet);
   }
   // getSongs() and addSong() methods
}
```

这个方法没有更新，所以它还是会把Song加到ArrayList中

创建参数化的HashSet来保存Song

addAll()可以复制其他集合的元素，效果就跟一个一个加进去一样

```
File Edit Window Help GetBetterMusic
%java Jukebox6

[Pink Moon, Somersault, Shiva Moon, Circles, Deep Channel,
Passenger, Listen, Listen, Listen, Circles]

[Circles, Circles, Deep Channel, Listen, Listen, Listen,
Passenger, Pink Moon, Shiva Moon, Somersault]

[Pink Moon, Listen, Shiva Moon, Circles, Listen, Deep Channel,
Passenger, Circles, Listen, Somersault]
```

排序前

排序后

放进HashSet之后列出

Set还是会有重复

已经设定的顺序在Set上消失了

对象要怎样才算相等？

首先我们得问一个问题：两个Song的引用怎样才算重复？它们必须被认为是相等的。这是说引用到完全相同的对象，还是有相同歌名的不同对象也算？

这带出一个很关键的议题：引用相等性和对象相等性。

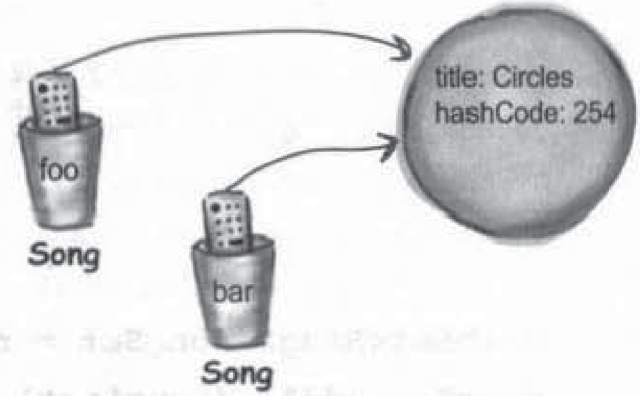
如果foo与bar两对象相等，则foo.equals(bar)会返回true，且两者的hashCode()也会返回相同的值。要让Set能把对象视为重复的，就必须让它们符合上面的条件被视为是相同的。

➤ 引用相等性

堆上同一对象的两个引用

引用到堆上同一个对象的两个引用是相等的。就这样。如果对两个引用调用hashCode()，你会得到相同的结果。如果没有被覆盖的话，hashCode()默认的行为会返回每个对象特有的序号（大部分的Java版本是依据内存位置计算此序号，所以不会有相同的 hashcode）。

如果想要知道两个引用是否相等，可以使用==来比较变量上的字节组合。如果引用到相同的对象，字节组合也会一样。



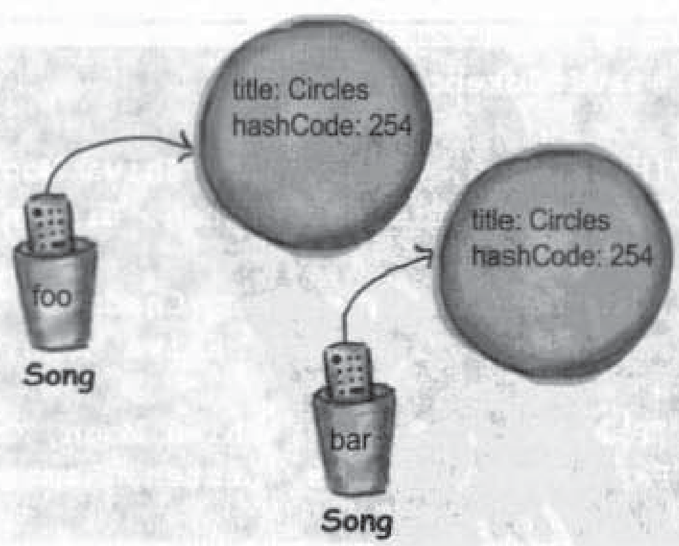

```
if (foo == bar) {
    // 两个引用都指向同一个对象
}
```

➤ 对象相等性

堆上的两个不同对象在意义上是相同的

如果你想要把两个不同的Song对象视为相等的，就必须覆盖过从Object继承下来的hashCode()方法与equals()方法。

就因为上面所说的内存计算问题，所以你必须覆盖过 hashCode( ) 才能确保两个对象有相同的 hashcode，也要确保以另一个对象为参数的equals()调用会返回true。




```
if (foo.equals(bar) && foo.hashCode() == bar.hashCode()) {
    // 两个引用指向同一个对象，或者两个对象是相等的
}
```

HashSet如何检查重复：hashCode()与equals()

当你把对象加入HashSet时，它会使用对象的hashcode值来判断对象加入的位置。但同时也会与其他已经加入的对象的hashcode作比对，如果没有相符的hashcode，HashSet就会假设新对象没有重复出现。

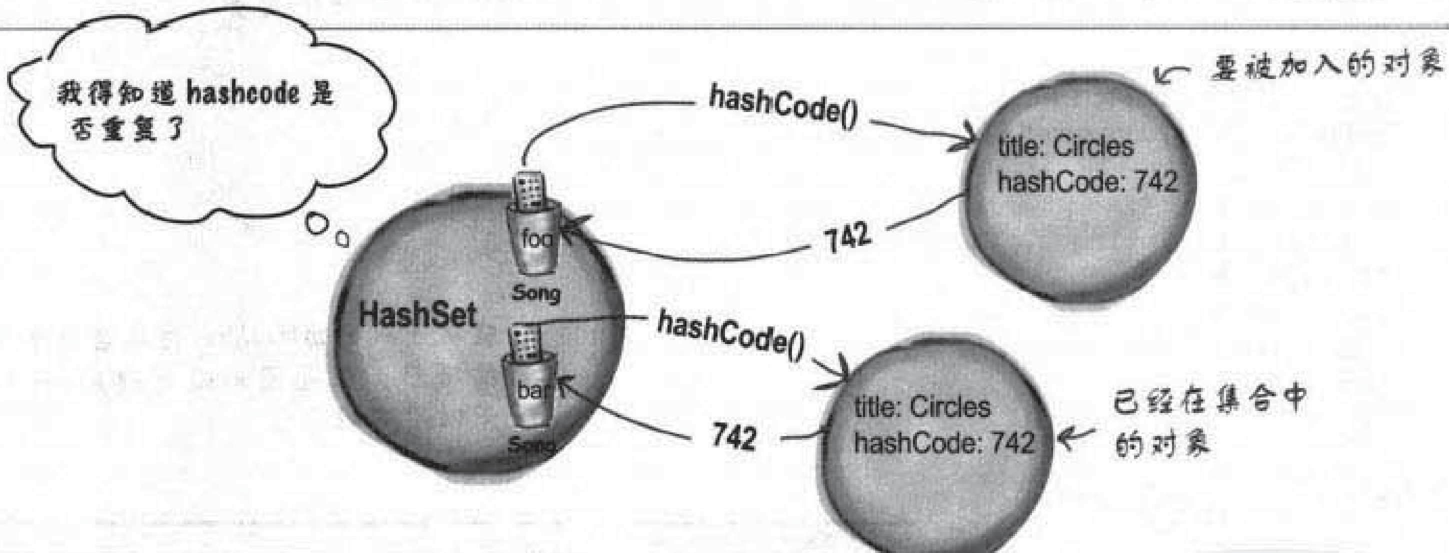
也就是说，如果hashcode是相异的，则HashSet会假设对象不可能是相同的。

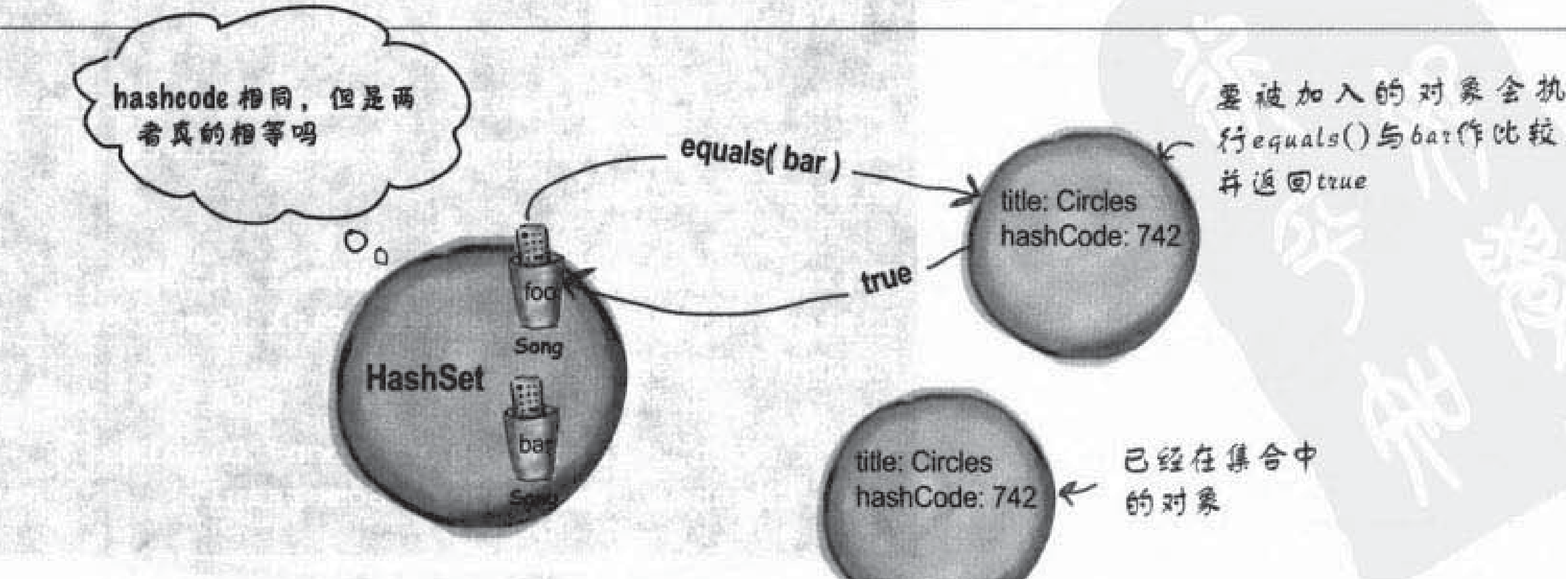
因此你必须override过hashCode()来确保对象有相同的值。

但有相同hashCode()的对象也不一定相等（下一页会讨论），所以如果HashCode找到相同hashcode的两个对象：新加入与本来存在的，它会调用其中一个的equals()来检查hashcode相等的对象是否真的相同。

如果两者相同，HashSet就会知道要加入的项目已经重复了，所以加入的操作就不会发生。

此时不会传回例外，但add()会返回以让你判别对象是否成功的加入。因此若add()返回false，就表示新对象与集合中的某项目被认为是重复的。





有覆盖过hashCode()与equals()的Song类

```
class Song implements Comparable<Song>{
   String title;
   String artist;
   String rating;
   String bpm;

  public boolean equals(Object aSong) {
     Song s = (Song) aSong;
     return getTitle().equals(s.getTitle());
  }

  public int hashCode() {
      return title.hashCode()
  }

   public int compareTo(Song s) {
      return title.compareTo(s.getTitle());
   }

   Song(String t, String a, String r, String b) {
      title = t;
      artist = a;
      rating = r;
      bpm = b;
   }

   public String getTitle() {
       return title;
   }

   public String getArtist() {
      return artist;
   }

   public String getRating() {
      return rating;
   }

  public String getBpm() {
    return bpm;
  }

  public String toString() {
    return title;
  }
}
```

要被比较的对象

因为歌名是String，且String本来就覆盖过的equals()，所以我们可以调用它

String也有覆盖过的hashCode()，注意到hashCode()与equals()使用相同的实例变量

成功了！列出HashSet时不会有重复的项目，但是因为没有调用sort()所以没有排序

```
File Edit Window Help RebootWindows
%java Jukebox6

[Pink Moon, Somersault, Shiva Moon, Circles,
Deep Channel, Passenger, Listen, Listen,
Listen, Circles]

[Circles, Circles, Deep Channel, Listen,
Listen, Listen, Passenger, Pink Moon, Shiva
Moon, Somersault]

[Pink Moon, Listen, Shiva Moon, Circles,
Deep Channel, Passenger, Somersault]
```

hashCode()与equals()的相关规定

API文件有对对象的状态制定出必须遵守的规则：

(1) 如果两个对象相等，则hashcode必须也是相等的。

(2) 如果两个对象相等，对其中一个对象调用equals()必须返回true。也就是说，若a.equals(b)则 b.equals(a)。

(3) 如果两个对象有相同的hashcode值，它们也不一定是相等的。但若两个对象相等，则hashcode值一定是相等的。

(4) 因此若equals()被覆盖过，则hashCode()也必须被覆盖。

(5) hashCode()的默认行为是对在heap上的对象产生独特的值。如果你没有override过hashCode()，则该class的两个对象怎样都不会被认为是相同的。

(6) equals()的默认行为是执行==的比较。也就是说会去测试两个引用是否对上heap上同一个对象。如果equals()没有被覆盖过，两个对象永远都不会被视为相同的，因为不同的对象有不同的字节组合。

a.equals(b)必须与
a.hashCode() == b.hashCode()等值。

但 a.hashCode() == b.hashCode()
不一定要与 a.equals()等值。

there are no Dumb Questions

问： 为什么不同对象会有相同hashcode的可能？

答： HashSet使用hashcode来达成存取速度较快的存储方法。如果你尝试用对象来寻找ArrayList中相同的对象（也就是不用索引来找），ArrayList会从头开始找起。但HashSet这样找对象的速度就快多了，因为它使用hashcode来寻找符合条件的元素。因此当你想要寻找某个对象时，通过hashcode就可以很快地算出该对象所在的位置，而不必从头一个一个找起。

数据结构的课程绝对会告诉你更完整的理论，但这样说明也能让你知道如何有效率地运用HashSet。事实上，如何开发出有效率的杂凑算法一直都是博士论文的热门题目。

重点在于hashcode相同并不一定保证对象是相等的，因为hashCode()所使用的杂凑算法也许刚好会让多个对象传回相同的杂凑值。越糟糕的杂凑算法越容易碰撞，但这也与数据值域分布的特性有关。总而言之，把数据结构这科目搞定就对了。

如果HashSet发现在比对的时候，同样的hashcode有多个对象，它会使用equals()来判断是否有完全的符合。也就是说，hashcode是用来缩小寻找成本，但最后还是要用equals()才能认定是否真的找到相同的项目。

如果想要保持有序，使用TreeSet

TreeSet在防止重复上面与HashSet是一样的。但它还会一直保持集合处于有序状态。如果使用TreeSet默认的构造函数，它工作起来就会像sort()一样使用对象的compareTo()方法来排序。但也可以选择传入Comparator给TreeSet的构造函数。缺点是如果不需要排序时就会浪费处理能力。但你可能会发现这点损耗实在很小。

```
import java.util.*;
import java.io.*;
public class Jukebox8 {
   ArrayList<Song> songList = new ArrayList<Song>();
   int val;

   public static void main(String[] args) {
      new Jukebox8().go();
   }

   public void go() {
      getSongs();
      System.out.println(songList);
      Collections.sort(songList);
      System.out.println(songList);
      TreeSet<Song> songSet = new TreeSet<Song>();
      songSet.addAll(songList);
      System.out.println(songSet);
   }

   void getSongs() {
      try {
         File file = new File("SongListMore.txt");
         BufferedReader reader = new BufferedReader(new FileReader(file));
         String line = null;
         while ((line= reader.readLine()) != null) {
              addSong(line);
         }

      } catch(Exception ex) {
         ex.printStackTrace();
      }
   }

   void addSong(String lineToParse) {
      String[] tokens = lineToParse.split("/");
      Song nextSong = new Song(tokens[0], tokens[1], tokens[2], tokens[3]);
      songList.add(nextSong);
   }
}
```

调用没有参数的构造函数来用TreeSet取代HashSet意味着以对象的compareTo()方法来进行排序

使用addAll()可以把对象全部加入

使用 TreeSet 一定要知道的事情……

它很容易使用，但你得先确定已经知道所有必须了解的事情。我们认为它太重要了，所以要让你通过习题来思考一下。先别翻到下一页，说真的！

Sharpen your pencil

仔细阅读这份程序代码，然后回答下面的问题（程序没有语法错误）。

```
import java.util.*;

public class TestTree {
   public static void main (String[] args) {
      new TestTree().go();
   }

   public void go() {
     Book b1 = new Book("How Cats Work");
     Book b2 = new Book("Remix your Body");
     Book b3 = new Book("Finding Emo");

     TreeSet<Book> tree = new TreeSet<Book>();
     tree.add(b1);
     tree.add(b2);
     tree.add(b3);
     System.out.println(tree);
   }
}

class Book {
   String title;
   public Book(String t) {
     title = t;
   }
}
```

(1) 编译会有什么结果？

____

(2) 如果能编译，执行TestTree会有什么结果？

____

(3) 如果有问题，要怎么更正？

____

TreeSet的元素必须是Comparable

TreeSet 无法猜到程序员的想法，你必须指出对象应该如何排序。

要使用TreeSet，下列其中一项必须为真。

- **集合中的元素必须是有实现Comparable的类型。**

上一页的Book并没有实现Comparable，所以执行时会有问题。想想看，可怜的TreeSet唯一的目的就是要保持元素有序，但是它根本不知道要如何排列Book对象！这在编译时不会有问题，因为TreeSet的add()并未要求取用Comparable类型的参数。如果你以TreeSet<Book>来创建，基本上add()只会要求元素得是Book，并没有检查Book是否有实现出Comparable！但加入第二个元素时，会因为无法调用对象的compareTo()而失败。

```
class Book implements Comparable {
   String title;
   public Book(String t) {
     title = t;
   }
   public int compareTo(Object b) {
     Book book = (Book) b;
     return (title.compareTo(book.title));
   }
}
```

或

- **使用重载、取用Comparator参数的构造函数来创建TreeSet。**

就像sort()，你可以选择使用元素的compareTo()，假设元素的类型都有实现Comparable，或者自定义Comparator。要使用自订的Comparator，你可以调用取用Comparator的构造函数。

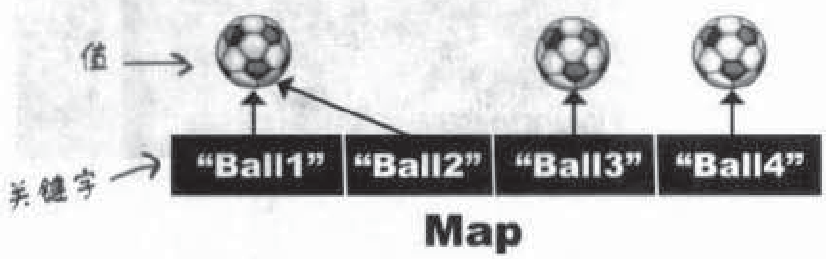
```
public class BookCompare implements Comparator<Book>{
   public int compare(Book one, Book two) {
      return (one.title.compareTo(two.title));
   }
}

class Test {
     public void go() {
     Book b1 = new Book("How Cats Work");
     Book b2 = new Book("Remix your Body");
     Book b3 = new Book("Finding Emo");
     BookCompare bCompare = new BookCompare();
     TreeSet<Book> tree = new TreeSet<Book>(bCompare);
     tree.add(new Book("How Cats Work");
     tree.add(new Book("Finding Emo");
     tree.add(new Book("Remix your Body");
     System.out.println(tree);
   }
}
```

现在来看Map

List和Set很好用，但有时Map才是最好的选择。

想象一下如果你需要用名称来取得值的情况。虽然key通常都是String，但也可以用任何Java的对象（或通过autoboxing的primitive）。



Map中的元素实际上是两个对象：关键字和值。值可以重复，但是key不行。

Map example

```
import java.util.*;

public class TestMap {

   public static void main(String[] args) {

      HashMap<String, Integer> scores = new HashMap<String, Integer>();

      scores.put("Kathy", 42);
      scores.put("Bert", 343);
      scores.put("Skyler", 420);

      System.out.println(scores);
      System.out.println(scores.get("Bert"));
   }
}
```

HashMap需要两个类型参数：关键字和值

使用put()取代add()，它需要两个参数

get()取用关键字参数，返回它的值

```
File Edit Window Help WhereAmI
%java TestMap

{Skyler=420, Bert=343, Kathy=42}
343
```

当你要列出Map时，它会以key=value的形式打印出，外面用{ }包起来

终于回到泛型

还记得本章较早之前讨论到取用泛型参数的方法会有多古怪吗？这是以多态的观点来说的。如果越看越觉得奇怪，就继续看下去，整个故事要好几页才说得完。

我们会先回顾一下数组参数是如何多态化运行的，然后再看同样的工作如何以泛型集合来完成。下面的程序代码在编译与执行时都不会有问题。

如果方法的参数是Animal的数组，它也能够取用Animal次类型的数组。

也就是说，如果方法是这样声明的：

void foo(Animal[] a) { }

若Dog有extend过Animal，你就可以用下列的两种方式调用：

foo(anAnimalArray);
foo(aDogArray);

普通数组工作的方式：

```
import java.util.*;

public class TestGenerics1 {
   public static void main(String[] args) {
      new TestGenerics1().go();
   }

   public void go() {
     Animal[] animals = {new Dog(), new Cat(), new Dog()};
     Dog[] dogs = {new Dog(), new Dog(), new Dog()};
     takeAnimals(animals);
     takeAnimals(dogs);
   }

  public void takeAnimals(Animal[] animals) {
     for(Animal a: animals) {
       a.eat();
      }
   }
}
```

声明并创建Animal数组，它带有Dog与Cat

声明并创建Dog数组，仅能带Dog

对两种数组调用takeAnimal()

重点在于takeAnimal()能够取用Animal[]或Dog[]参数，因为Dog是一个Animal，多态在此发挥作用

记得吗？我们只能调用声明在Animal中的方法，因为它的参数是Animal数组，且无需任何类型转换（怎么转换？数组可能同时带有Dog与Cat）

```
abstract class Animal {
   void eat() {
     System.out.println("animal eating");
   }
}
class Dog extends Animal {
   void bark() { }
}
class Cat extends Animal {
   void meow() { }
}
```

简单版的Animal继承层次

使用多态参数与泛型

我们已经看过数组的整个工作状况，但当我们把array换成ArrayList时还会一样吗？听起来很合理，不是吗？

先来看只有Animal的ArrayList的状况。我们仅对go()作修改。

只传入ArrayList<Animal>

从Animal[ ]改成ArrayList<Animal>

```
public void go() {
    ArrayList<Animal> animals = new ArrayList<Animal>();
    animals.add(new Dog());
    animals.add(new Cat());
    animals.add(new Dog());

    takeAnimals(animals);
}
```

只能一个一个加进去，没有像数组的那种语法可用

除了animals这个变量引用ArrayList而不是数组之外，程序代码都相同

```
public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {
        a.eat();
    }
}
```

此方法改成取用ArrayList，其余都一样，for循环可以操作集合与数组

可编写可运行，没有问题

```
File Edit Window Help CatFoodIsBetter
%java TestGenerics2
animal eating
animal eating
animal eating
animal eating
animal eating
animal eating
```

但 ArrayList<Dog> 可用吗?

因为多态的关系，编译器会让Dog数组通过取用Animal数组参数的方法。没问题，且ArrayList<Animal>也可以通过取用ArrayList<Animal> 的方法。问题是，ArrayList<Animal>参数能够接受ArrayList<Dog>吗？如果在数组上是可以的，这应该要也可以吧？

只传入ArrayList<Dog>

```
public void go() {
   ArrayList<Animal> animals = new ArrayList<Animal>();
   animals.add(new Dog());
   animals.add(new Cat());
   animals.add(new Dog());
   takeAnimals(animals); ←— 我们知道这里不会有问题

   ArrayList<Dog> dogs = new ArrayList<Dog>();
   dogs.add(new Dog());
   dogs.add(new Dog());            创建Dog的ArrayList
   takeAnimals(dogs); ←— 改成ArrayList之后还会过关吗?
}

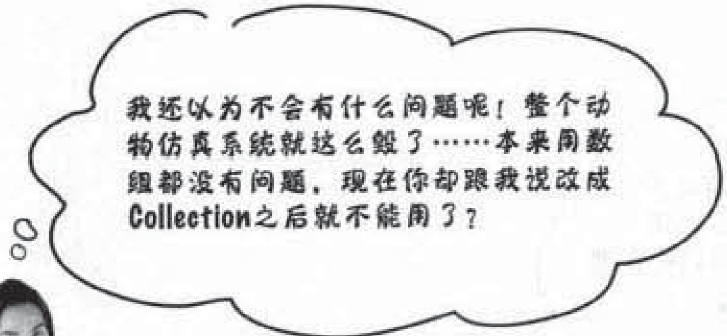
public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {
    a.eat();
   }
}
```

编译时：

```
File Edit Window Help CatsAreSmarter
%java TestGenerics3

TestGenerics3.java:21: takeAnimals(java.util.
ArrayList<Animal>) in TestGenerics3 cannot be applied to
(java.util.ArrayList<Dog>)
     takeAnimals(dogs);
     ^
1 error
```

想不到啊，看起来都没有问题……





如果可以会怎样？

假设编译器让你过关。它允许对下面这个方法传入ArrayList<Dog>作为参数：

```
public void takeAnimals(ArrayList<Animal> animals) {
    for(Animal a: animals) {
        a.eat();
    }
}
```

程序看起来不太像是会有问题的样子，对吧？毕竟多态的意义在于Animal可以做的事情（此列中的eat()），Dog也都能做。所以对Dog引用调用eat()会出什么状况呢？

不会，当然不会。

上面的程序代码没问题，但下面这段程序代码：

```
public void takeAnimals(ArrayList<Animal> animals) {
    animals.add(new Cat());   ← 加入一只猫
}
```

这就会是问题了。理论上把Cat加到ArrayList<Animal>是合法的，这也是使用 Animal 的本意——让各种 Animal 都可以加入到此ArrayList中。

但如果你传入一个Dog的ArrayList给该方法，那么结果就是一个被Cat给混进去的Dog集合。原本不可能出现Cat的ArrayList<Dog>在这种情况下还是出问题了，这就是为什么编译器不会让它通过的原因。

如果把方法声明成取用ArrayList<Animal>，它就只会取用ArrayList<Animal>参数，ArrayList<Dog>与ArrayList<Cat>都不行。



数组的类型是在运行期间检查的，但集合的类型检查只会发生在编译期间

假设你对Dog[]加入一个Cat（把Dog[]传入声明为Animal[]的参数中是完全合法的操作）

```
public void go() {
   Dog[] dogs = {new Dog(), new Dog(), new Dog()};
   takeAnimals(dogs);
}

public void takeAnimals(Animal[] animals) {
   animals[0] = new Cat();
}
```

把Cat放到Dog数组中，这能骗过编译器

可以编译，但运行的时候：

哇！被Java虚拟机发现了！

File Edit Window Help CatsAreSmarter

```
%java TestGenerics1
Exception in thread "main" java.lang.ArrayStoreException:
Cat
        at TestGenerics1.takeAnimals(TestGenerics1.java:16)
        at TestGenerics1.go(TestGenerics1.java:12)
        at TestGenerics1.main(TestGenerics1.java:5)
```


如果有办法能够使用多态化集合参数该多好，这样我就能够传入小猫或小狗的集合了。最好是还能调用每个元素的eat()但又不允许把霸青猫加到一群可鲁中……算了。我还是继续看我的文艺爱情火爆动作故事书吧！

万用字符

这听起来很怪，但却还有一种能够创建出接受Animal子型参数的方法，就是使用万用字符（wildcard），这是它被加入Java语言中的原因。

```
public void takeAnimals(ArrayList<? extends Animal> animals) {
    for(Animal a: animals) {
        a.eat();
    }
}
```

要记住此处的extends同时代表继承和实现。如果要取用有实现Pet这个接口类型的ArrayList，也是这样声明：

ArrayList<? extends Pet>

你可能会怀疑“有什么差别？问题还不是一样？Dog群还不是有可能被加入Cat”。

你的疑问没错，但实际上在使用带有<?>的声明时，编译器不会让你加入任何东西到集合中！

在方法参数中使用万用字符时，编译器会阻止任何可能破坏引用参数所指集合的行为。

你能够调用 list 中任何元素的方法，但不能加入元素。

也就是说，你可以操作集合元素，但不能新增集合元素。如此才能保障执行期间的安全性，因为编译器会阻止执行期的恐怖行动。

所以下面这个程序是可以的：

```
for(Animal a: animals) {
        a.eat( );
}
```

但这个就过不了编译：

```
animals.add(new Cat( ));
```

相同功能的另一种语法

你或许还记得当讨论到sort()方法时，它使用到泛型类型，但以一个不寻常的格式在返回类型前声明类型参数。这是另外一种类型参数的声明方式，但结果是一样的。

这一行：

```
public <T extends Animal> void takeThing(ArrayList<T> list)
```

跟这一行执行一样：

```
public void takeThing(ArrayList<? extends Animal> list)
```

there are no Dumb Questions

问： 如果都一样，为什么要用有问号的那个？

答： 这要看你是否会使用到T来决定。举例来说，如果 方法有两个参数——都是继承Animal的集合会怎样？此时，只声明一次会比较有效率。

```
public <T extends Animal> void takeThing(ArrayList<T> one, ArrayList<T> two)
```

而不必这样：

```
public void takeThing(ArrayList<? extends Animal> one,
                                        ArrayList<? extends Animal> two)
```



我是编译器



你的任务是扮演编译器角色并判断下列哪些程序是可以通过编译的。有些程序代码并没有被讨论过，所以要根据你已知的规则来回答。也有可能需要用猜的，但要猜得有根据。

可以通过编译吗？

```
❑ ArrayList<Dog> dogs1 = new ArrayList<Animal>();

❑ ArrayList<Animal> animals1 = new ArrayList<Dog>();

❑ List<Animal> list = new ArrayList<Animal>();

❑ ArrayList<Dog> dogs = new ArrayList<Dog>();

❑ ArrayList<Animal> animals = dogs;

❑ List<Dog> dogList = dogs;

❑ ArrayList<Object> objects = new ArrayList<Object>();

❑ List<Object> objList = objects;

❑ ArrayList<Object> objs = new ArrayList<Dog>();
```

逆向工程解答

```
import java.util.*;

public class SortMountains {

  LinkedList<Mountain> mtn = new LinkedList<Mountain>();

  class NameCompare implements Comparator <Mountain> {
    public int compare(Mountain one, Mountain two) {
      return one.name.compareTo(two.name);
    }
  }
  class HeightCompare implements Comparator <Mountain> {
    public int compare(Mountain one, Mountain two) {
      return (two.height - one.height);
    }
  }
  public static void main(String [] args) {
    new SortMountain().go();
  }
  public void go() {
    mtn.add(new Mountain("Longs", 14255));
    mtn.add(new Mountain("Elbert", 14433));
    mtn.add(new Mountain("Maroon", 14156));
    mtn.add(new Mountain("Castle", 14265));

    System.out.println("as entered:\n" + mtn);
    NameCompare nc = new NameCompare();
    Collections.sort(mtn, nc);
    System.out.println("by name:\n" + mtn);
    HeightCompare hc = new HeightCompare();
    Collections.sort(mtn, hc);
    System.out.println("by height:\n" + mtn);
  }
}

class Mountain {
  String name;
  int height;

  Mountain(String n, int h) {
    name = n;
    height = h;
  }
  public String toString( ) {
    return name + " " + height;
  }
```

你注意到这是降幂排序吗?

输出:

```
File Edit Window Help ThisOne'sForBob
%java SortMountains
as entered:
[Longs 14255, Elbert 14433, Maroon 14156, Castle 14265]
by name:
[Castle 14265, Elbert 14433, Longs 14255, Maroon 14156]
by height:
[Elbert 14433, Castle 14265, Longs 14255, Maroon 14156]
```

填空题解答

可用的答案：

Comparator,

Comparable,

compareTo( ),

compare( ),

yes,

no

对下面这行程序来说：

```
Collections.sort(myArrayList);
```

(1) 存储在myArrayList中对象的类一定要实现什么？	Comparabl
(2) 存储在myArrayList中对象的类一定要实现什么方法？	compareTo()
(3) 存储在myArrayList中对象的类能够同时实现Comparator与Comparable吗？	yes

对下面这行程序来说：

```
Collection.sort(myArrayList, myCompare);
```

(4) 存储在myArrayList中对象的类能够实现Comparable吗？	yes
(5) 存储在myArrayList中对象的类能够实现Comparator吗？	yes
(6) 存储在myArrayList中对象的类必须实现Comparable吗？	no
(7) 存储在myArrayList中对象的类必须实现Comparator吗？	no
(8) myCompare对象的类必须实现什么？	Comparator
(9) myCompare对象的类必须实现什么方法？	compare()



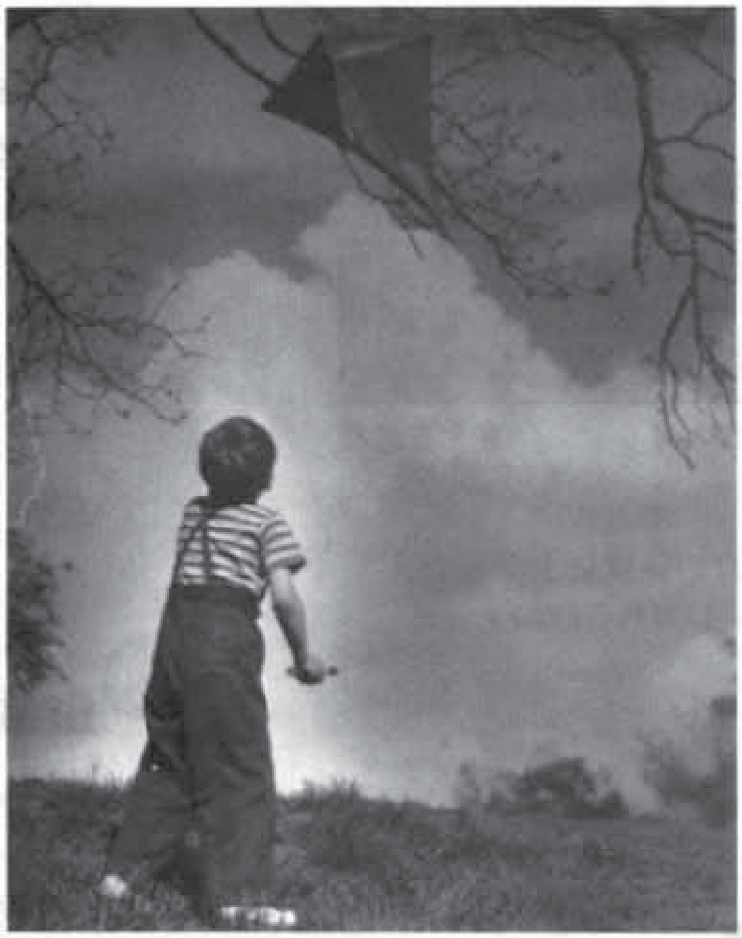
我是编译器

可以通过编译吗？

- [ ] `ArrayList<Dog> dogs1 = new ArrayList<Animal>();`
- [ ] `ArrayList<Animal> animals1 = new ArrayList<Dog>();`
- [x] `List<Animal> list = new ArrayList<Animal>();`
- [x] `ArrayList<Dog> dogs = new ArrayList<Dog>();`
- [ ] `ArrayList<Animal> animals = dogs;`
- [x] `List<Dog> dogList = dogs;`
- [x] `ArrayList<Object> objects = new ArrayList<Object>();`
- [x] `List<Object> objList = objects;`
- [ ] `ArrayList<Object> objs = new ArrayList<Dog>();`

17 包、jar存档文件和部署

发布程序



该是放手的时候了。你写程序、测试、修改。你告诉每个人你想要转行去开出租车。最后，你还是写出可以出国比赛的好程序——它居然可以运行！那现在呢？如何交到用户的手上？到底要给客户什么东西？如果不知道谁会用呢？最后这两章会来讨论如何组织、包装与部署Java程序。我们会触及包括可执行的jar、Java Web Start、RMI与Servlets等本机、半本机与远程部署的选项。这一章大部分的内容会叙述程序代码的组织与包装——不管最后标的是什么都得知道的事情。放心，发布程序并不是分手，而是没完没了的维护工作的开始……

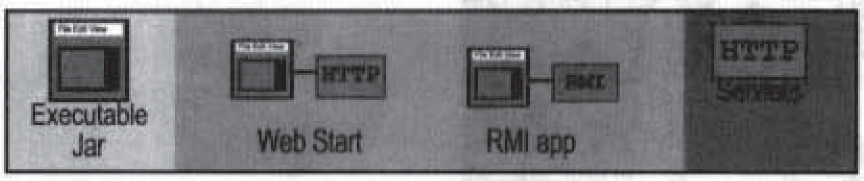
部署应用程序

到底什么才是Java应用程序？也就是说开发完成之后，要交付的项目是什么？用户的系统很可能跟你的不一样。更重要的是，他们并没有拿到应用程序。所以现在该把你的程序塑造成可部署给外人使用的形式。在这一章中，我们会讨论本机部署，包括Executable Jar与称为Java Web Start的半本机半远程技术。下一章会讨论较远程的部署选择，包括RMI和Servelet在内。

Java程序是由一组类所组成。那就是开发过程的输出。

真正的问题是完成之后要拿这些类怎么办？

部署的选择



完全在本机　　介于两者之间　　完全在远程

① **本机。**
整个程序都在用户的计算机上以独立、可携的GUI执行，并以可执行的Jar来部署（稍后讨论JAR）。

② **两者之间的组合。**
应用程序被分散成在用户本地系统运行的客户端，连接到执行应用程序服务的服务器部分。

③ **远程。**
整个应用程序都在服务器端执行，客户端通过非Java形式、可能是浏览器的装置来存取。

Brain Barbell

把应用程序部署成在客户端计算机上独立执行的程序有什么好处和坏处？

把应用程序部署成在独立在服务器上执行的Web程序、并让用户通过浏览器交互有什么好处和坏处？

但在我们开始进入关于部署的内容之前，先回头来看看当你完成程序时，你只把类文件交给用户会发生什么事。工作目录中到底有什么东西？



想象这个场景……

大傻正在愉快地进行Java程序最后的部分，经历好几个礼拜“就快写好了”的状态，这次真地就要写完了。这只程序是相当复杂的GUI程序，但因为有Swing的帮助，他实际上只需要写出9个类。

最后，交付给客户的时间到了。他发现只要把那9个类文件拷贝给客户就行，因为客户端已经安装好了Java API。他切换到放文件的目录开始**ls**……



啊!目录下面不只有18个文件（9个源代码、9个编译出的类），实际上有31个文件，好几个文件的名字怪怪的。

Account$FileListener.class

Char$SaveListener.class

类似这样。他完全忘记编译器还必须对内部类的GUI事件监听程序产生出类文件，就是那些怪怪的文件。

现在他小心地把文件抽出来，少了一个文件程序就不能运行。但他又不想不小心把源代码交给客户，并且整个目录看起来也很乱。

将源代码与类文件分离

带有一堆源代码和类文件的目录是一团混乱的。大傻应该要好好地整理一下文件，让源代码与编译出的文件分开。也就是说，确保编译过的类文件不会放在源代码的目录中。

关键在于结合-d这个编译选项和目录组织的结构。

有好几种方法可以组织文件，你们公司可能有规定要怎么做。然而我们会建议一种几乎已成为标准的组织化纲要。

使用这种纲要时，你会创建出项目目录，下面有source和classes目录。把源代码（.java）存储在source目录下。在编译时动点手脚让输出（.class）产生在classses目录。

有个编译器选项能够这么搞。

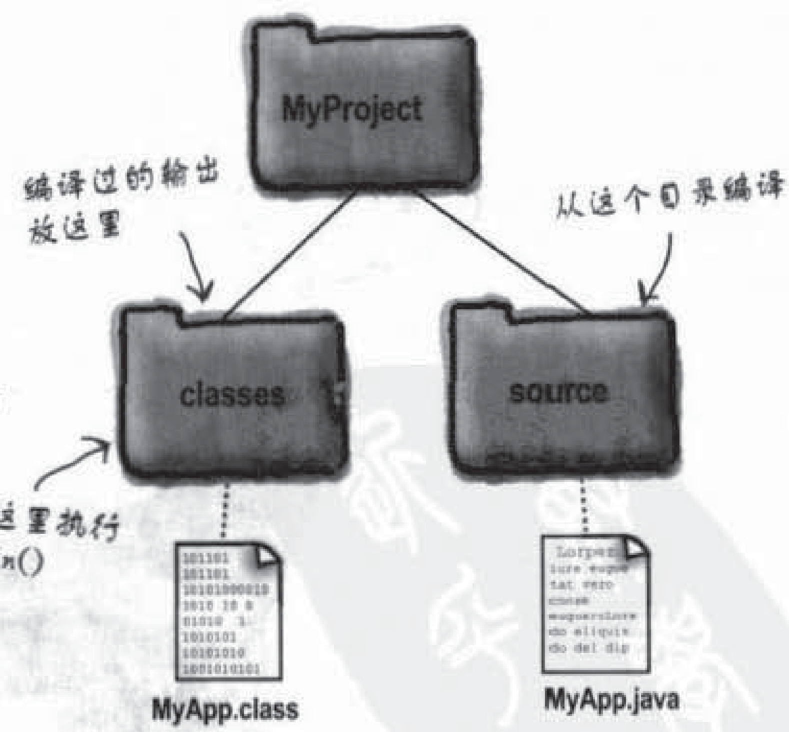
编译时加上**-d**（directory）选项。




```
%cd MyProject/source
%javac -d ../classes MyApp.java
```

要求编译器将编译输出放到目前目录上一层下面的classes目录

这里放的还是要被编译的文件



使用-d选项，你就可以指定编译过的程序要摆在哪里，而不会放到默认的同一个目录下。若要编译全部的.java文件：

```
%javac -d ../classes *.java
```

代表目前目录所有的源文件

执行程序

```
%cd MyProject/classes
%java MyApp
```

从classes目录来运行

（除错信息：这一章假设目前目录是在classpath中，如果你修改过环境变量，要确定“.”包含在classpath中。）

把程序包进JAR



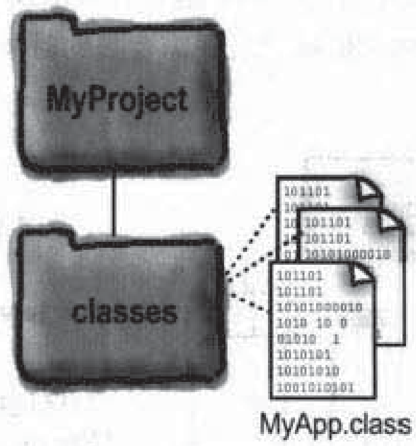
JAR就是Java ARchive。这种文件是个pkzip格式的文件，它能让你把一组类文件包装起来，所以交付时只需要一个JAR文件。如果你很熟悉UNIX上的tar命令的话，你就会知道jar这个工具要怎么使用（注意：当我们提到全大写的JAR时是说集合起来的文件，全小写的jar是用来整理文件的工具）。

问题是用户要拿JAR怎么办？

你会创建出可执行的JAR。

可执行的JAR代表用户不需把文件抽出来就能运行。程序可以在类文件保存在JAR的情况下执行。秘诀在于创建出manifest文件，它会带有JAR的信息，告诉Java虚拟机哪个类含有main()这个方法！

创建可执行的JAR



① 确定所有的类文件都在classes目录下

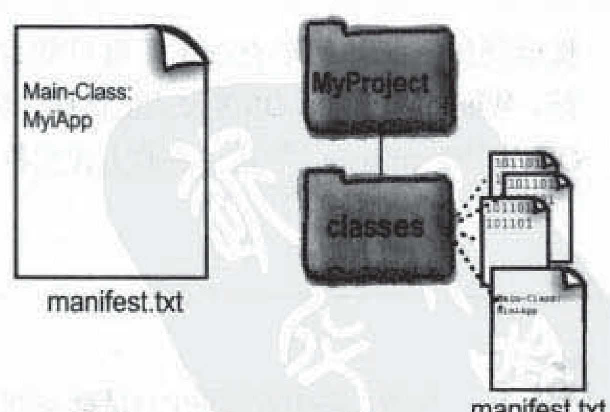
稍后还有进一步的讨论，但现在先把所有的类文件放在classes目录下。

② 创建manifest.txt来描述哪个类带有main()方法

该文件带有下面这一行：

Main-Class: MyApp ← 后面没有.class

在此行后面要有换行，否则有可能出错。将此文件放在classes目录下。

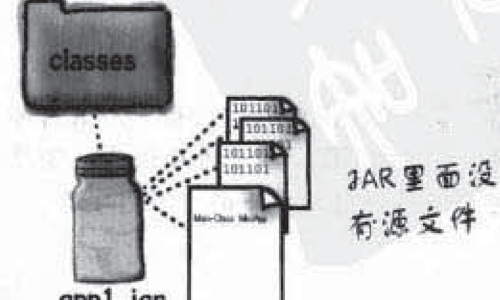


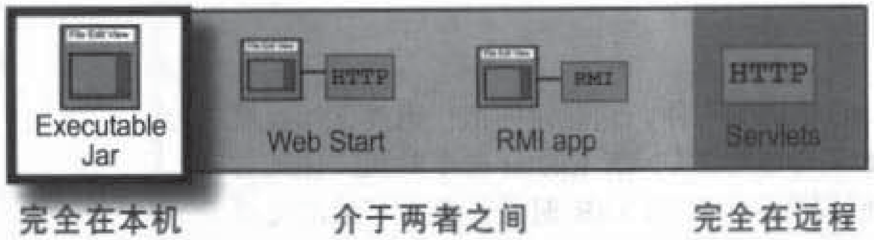
③ 执行jar工具来创建带有所有类以及manifest的JAR文件

```
%cd MiniProject/classes
%jar -cvmf manifest.txt app1.jar *.class
```

或

```
%jar -cvmf manifest.txt app1.jar MyApp.class
```

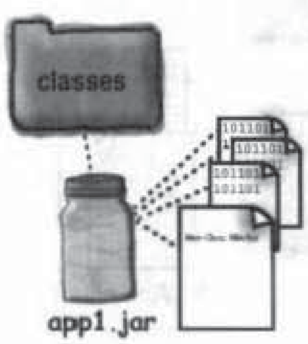




大部分完全在本机的Java应用程序都是以可执行的JAR来部署的。

执行JAR

Java虚拟机能够从JAR中载入类，并调用该类的main()方法。事实上，整个应用程序都可以包在JAR中。一旦main()方法开始执行，Java虚拟机就不会在乎类是从哪里来的，只要能够找到就行。其中一个来源就是classpath指定位置的所有JAR文件。如果看到某个JAR，则Java虚拟机就会在需要类的时候查询此 JAR。




```
%cd MyProject/classes

%java -jar app1.jar
```

Java虚拟机必须要能找到JAR，所以它必须在classpath下。让JAR曝光的最好方式就是把JAR放在工作目录下。

-jar标识告诉Java虚拟机所给的是个JAR

Java虚拟机会检查JAR的manifest寻找入口，如果没有就会发生运行期间异常

根据操作系统如何动态设定，有可能直接双击JAR就可以开始执行，Windows与Mac OS X大致是这样。你可以通过点选JAR并要求OS以“Open with...”这一类的方式来打开。

there are no Dumb Questions

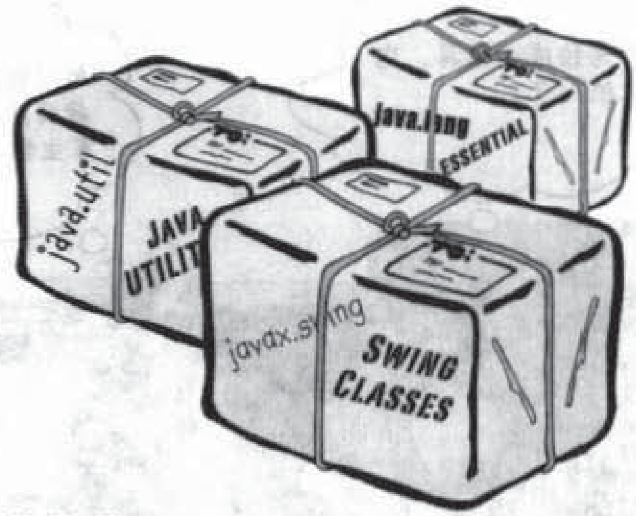
问： 为什么不干脆把整个目录都 JAR 掉？

答： Java虚拟机会检查JAR内部并预期找到所需的东西。它不会深入其他的目录找，除非类是包的一部分，或者目录符合包指令下的？万用字符。

问： 你说什么？

答： 你不能把类文件放进某个目录后就这样包起来。但如果类属于某个包，那么你就可以把整个包结构给JAR起来。事实上是必须这么做的。稍后会对这做出解释。

把类包进包中！



写出可重复使用的类时，你会把它们放到内部的函数库给其他的程序员使用。就在你谦虚地让大家使用这些面向对象完美经典范例时，电话响了。一种不祥的电话。大傻也在函数库里面加进了相同名称的类，因此该出的状况一样也没少，你遇上了命理学所谓的姓名相克状况。

这都要怪你没有使用包！其实你使用包，把Java API包起来的那些包。但你没有把自己写的类包进包中，实际上来说这实在很糟糕。

我们要把前几页展示的组织结构做个修改，让类包进包中，然后再把整个包给JAR起来。

接下来的内容要专心注意，细微的细节差异就可能让程序无法编译或执行。

Java API有这些包结构：

java.text.NumberFormat

java.util.ArrayList

java.awt.FlowLayout

java.awt.event.ActionEvent

java.net.Socket

用包防止类名称的冲突



虽然包不只是用来防止名称冲突，但这也是主要的目的之一。你可能会写出称为Customer、Account与ShoppingCart的类。当然啦，超过一半以上写电子商务应用程序的开发者也会对class取这种菜市场名。这对面向对象来说是很危险的。如果面向对象的重点之一是写出可以重用的组件，开发者就必须要能够组合各种来源的组件产生新的应用。你的组件也要能够跟其他组件和平相处，包括那些从你根本没想到的来源所写的。

这看起来像什么？是不是很像文件目录结构？

回忆一下第6章我们讨论过包的名称就像是类的全名，技术上称为fully-qualified name。ArrayList其实是java.util.ArrayList，JButton其实叫做javax.swing.JButton，而Socket全名是java.net.Socket。注意到其中两个都是以java开头的。当你在处理包结构时要想到继承层次，并把你的类也做同样的安排。



包可以防止名称冲突，但这只会在包名称保证不会重复的情况下起作用。最好的方式是在前面加上domain名称。

防止包命名冲突

把类包进包中可以减少与其他类产生命名冲突的机会，但要如何防止两个程序员做出同名的包呢？也就是说如何避免大家都把Account这个类放进名称为shopping.customers的包呢？如此一来最后的名称也会一样：

shopping.customers.Accoun

Sun 建议的命名规则能够大幅降低冲突的可能性——加上你所取得的域名称。它会是独一无二的。也许有好几个人都会叫做“淑芬”，但是不会有两个网域都叫“oreilly.com.cn”。

com.headfirstbooks.Book

包名称

类名称

反向使用domain的包名称

com.headfirstjava.projects.Chart

将domain名称反过来放在前面

类的名称第一个字母是大写的

这个名字也许很常见，加上domain之后就只需担心同公司的人

把类包进包中

① 选择包名称

我们以com.headfirstjava为例。此类的名称为PackageExercise，因此完整的名称会是：com.headfirstjava.PackageExercise。

② 在类中加入包指令

这必须是程序源文件的第一个语句，比import语句还要靠上。每个原始文件只能有一个包指令，因此同一文件中的类都会在同一个包中。当然也包括内部。

```
package com.headfirstjava;

import javax.swing.*;

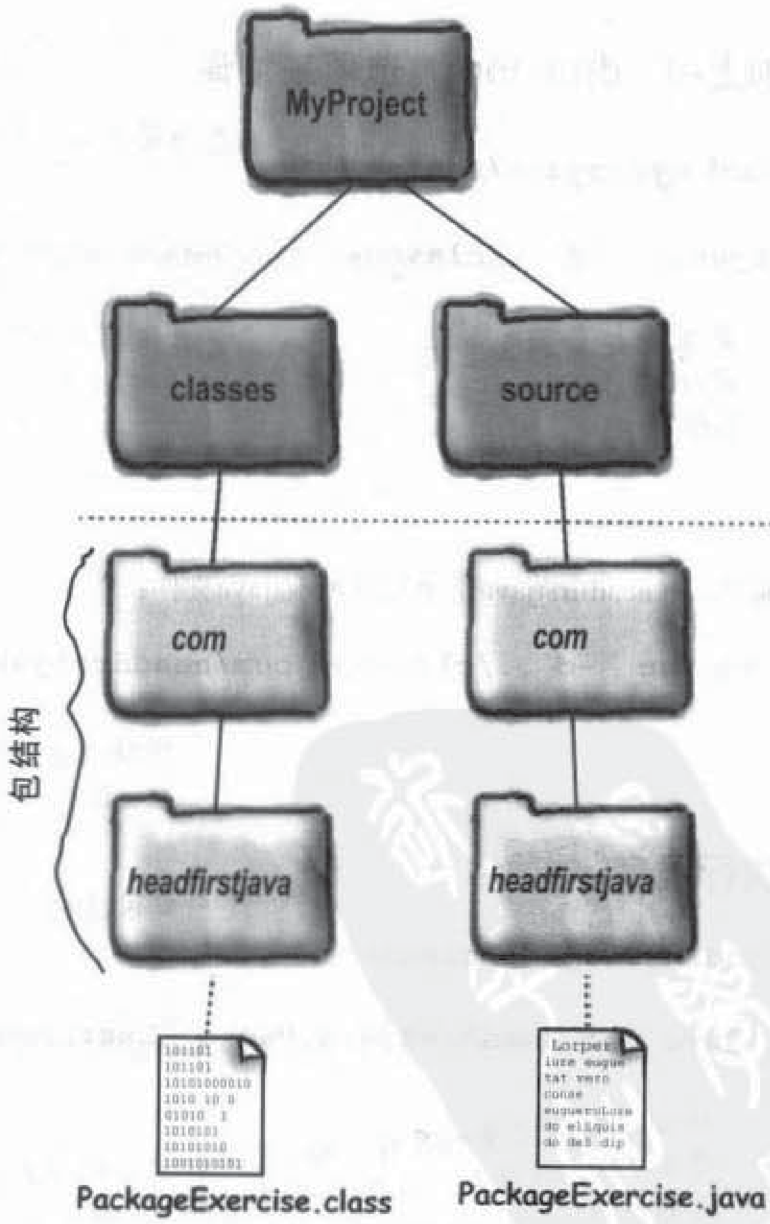
public class PackageExercise {
   // life-altering code here
}
```

③ 设定相对应的目录结构

只是把包指令加入源文件是不够的。类不会真的被加入包中，除非类也在相对应的目录结构中。因此，如果完整名称是com.headfirstjava.PackageExercise，则你必须把PackageExercise源文件放在名为headfirstjava的目录下，此目录必须在com目录下。

不这样做也可以编译，但相信我们，你不会想找自己的麻烦。让源代码摆在相对应的目录下你就可以避免令人头痛的问题。

你必须把类放在与包层次结构相以应的目录结构下。



将源文件和类都做出相对应的结构

编译与执行包

当类包在包中，编译与执行都要有点技巧。主要的问题来自于编译器和Java虚拟机都得要能够找到你的类以及所用到的其他类。对于核心API的类来说这不是问题。Java一定会知道它们在哪里。但对你的类而言，从单一目录来编译源文件是不可能的（至少是不可靠的）。然而我们保证如果能够照着这一页所描述的方法进行，你就一定会成功。还有其他的方法也可行，但我们发现这是最可靠也最容易遵循的方式。

加上-d（directory）选项来编译

```
%cd MyProject/source
```

呆在当前目录，别切换到.java文件的目录

```
%javac -d ../classes com/headfirstjava/PackageExercise.java
```

要求编译器将输出放到class目录下正确的包位置上

现在得指定取得源文件的路径

编译com.headfirstjava这个包的所有.java文件：

```
%javac -d ../classes com/headfirstjava/*.java
```

编译此目录下所有的.java文件

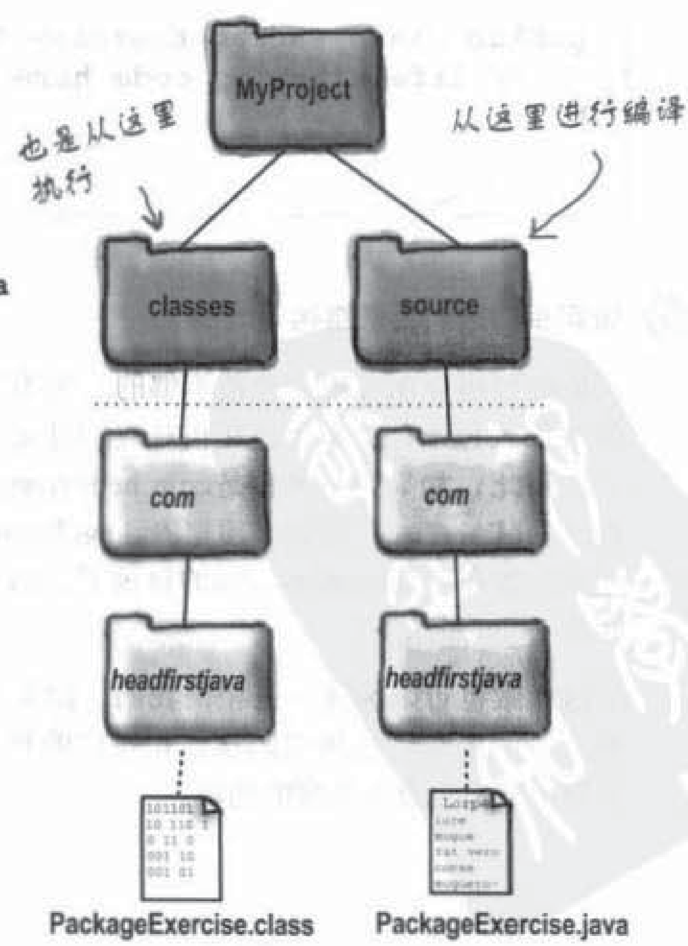
执行程序

从classes目录执行

```
%cd MyProject/classes
```

```
%java com.headfirstjava.PackageExercise
```

你必须指定完整的名称！Java虚拟机会看得懂并找寻当前目录下的com目录，其下应该有headfirstjava目录，那里应该能找到class。class在其他位置都无法运行！

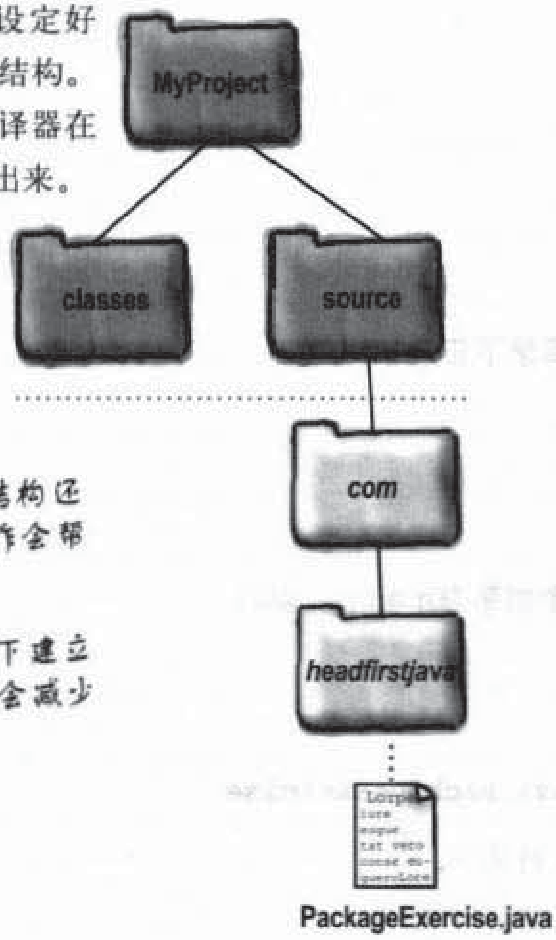


其实-d选项真地很酷

加上-d来编译是很棒的事情，因为它不仅让你把编译结果输出到别的地方，它还可以把类依照包的组织放到正确的目录上。

更棒的还在后头！

假设说你已经把源代码的目录结构设定好了，但还没有设定相对应的输出目录结构。没问题！加上-d的编译操作也会让编译器在遇到目录结构尚未建立时主动帮你做出来。



如果classes目录下包的目录结构还没有建好，使用-d的编译操作会帮你照顾好

因此你无需事先在classes之下建立目录结构，实际上这样反而会减少打字错误的可能

-d选项会要求编译器将编译结果根据包的结构来建立目录并输出，如果目录还没有建好，编译器会自动地处理这些工作。

there are no Dumb Questions

问： **我切换到主文件的目录下面，结果Java虚拟机就说找不到类！可是文件明明就在当前目录啊！**

答： 一旦类被包进包中，你就不能用"简写"的名称来调用它。你必须在命令栏指定要执行main()的类的完整名称，这包括了包结构，Java会坚持类必须要待在相对应的目录结构下。若命令栏的指令像这样：

```
%java com.foo.Book
```

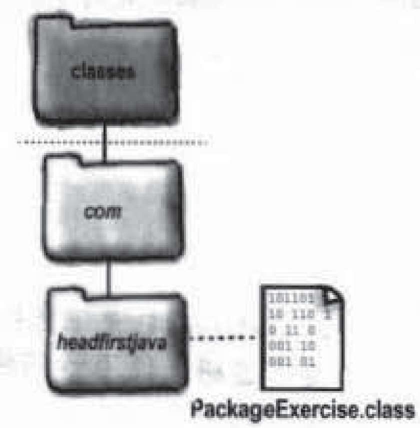
则Java虚拟机会从当前目录寻找名称为com的目录。在还没有找到带有foo子目录的com目录之前它不会去找寻名称为Book的类。只有在该目录下找到的Book才会被Java虚拟机接受。Java虚拟机也不会往上观察目录名称刚好是com然后决定就在这里找。

以包创建可执行的JAR



当你把类包进包中，包目录结构必须在JAR中！你不能只是把类装到JAR里面，还必须确定目录结构没有多往上走。包的第一层目录（通常是com）必须是JAR的第一层目录！如果你不小心从上面的目录包下来（例如从classes开始包），JAR就无法正确运行。

我创建可执行的JAR



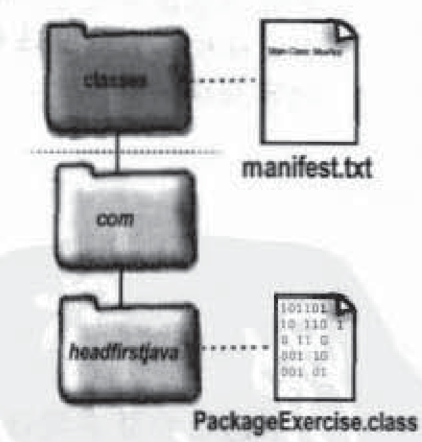
① 确定所有的类文件都放在class目录下正确相对应的包结构中

② 创建manifest.txt文件来描述哪个类带有main()，以及确认有使用完整的类名称

在 manifest.txt 写入一行：

```
Main-Class: com.headfirstjava.PackageExercise
```

然后把 manifest 文件放到 classes 目录下。



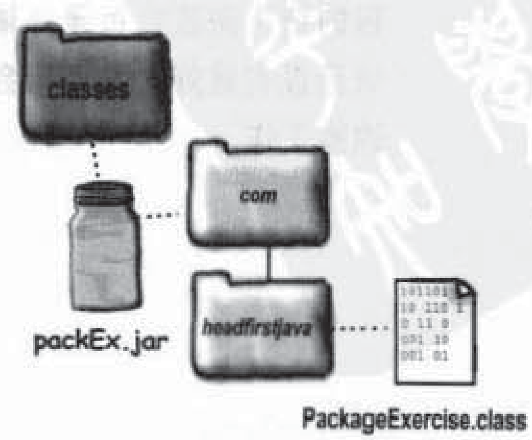
③ 执行jar工具来创建带有目录结构与manifest的JAR文件

只要从com开始就行，其下整个包的类都会被包进去JAR。

```
%cd MyProject/classes

%jar -cvmf  manifest.txt  packEx.jar  com
```

只要指定com目录就行，剩下都不会有问题

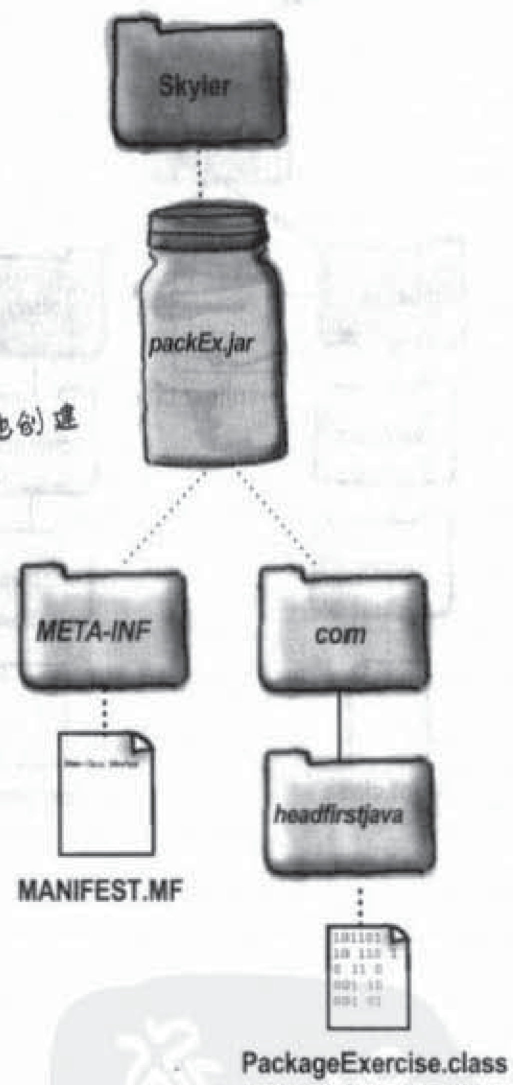


那manifest文件会跑到哪里？

看进去JAR就会知道。jar工具不只可以从命令栏中创建和执行JAR而已，你也可以把JAR的内容物解压出来（就像unzip或untar一样）。

假设说你已经把packEx.jar放到Skyler这个目录下。

把JAR文件放到Skyler目录下



jar工具会自动地创建META-INF目录

条列和解压的jar命令

① 将JAR内容列出。

```
% jar -tf packEx.jar
```

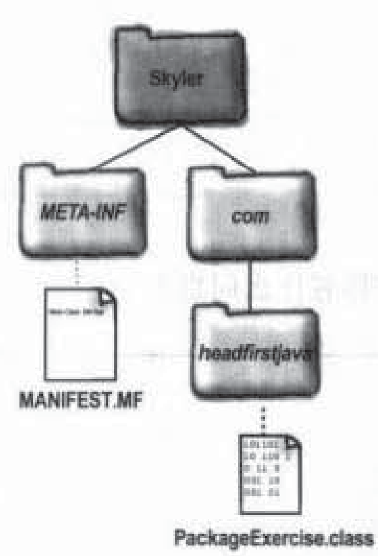
tf代表Table File，也就是列出文件的列表。

```
File Edit Window Help Pickle
% cd Skyler
% jar -tf packEx.jar
META-INF/
META-INF/MANIFEST.MF
com/
com/headfirstjava/
com/headfirstjava/PackageExer
cise.class
```

② Extract the contents of a JAR (i.e. unjar)

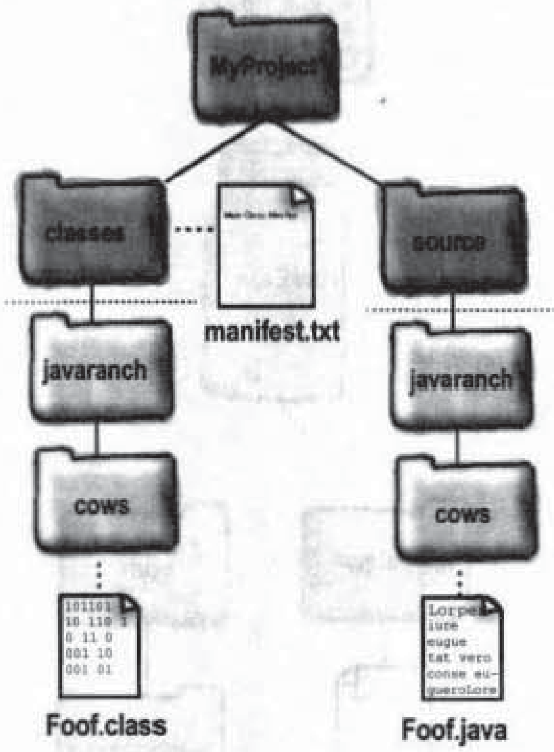
```
% cd Skyler
% jar -xf packEx.jar
```

xf代表eXtract File，就像unzip一样，如果把packEx.jar解开，你会在当前目录之下看到META-INF和com目录



META-INF代表META INFormation，jar工具会自动创建出这个目录和MANIFEST.MF文件，你的manifest.txt不会被带进JAR中，但它的内容会放进真正的manifest中。

Sharpen your pencil



观察左图表示的包目录结构，写出应该在命令栏输入什么样的指令才能编译、执行、创建JAR、执行JAR。假设包目录结构是以标准的source与classes布局方式设定的。

编译：

```
%cd source
%javac ______________________
```

执行：

```
%cd __________
%java ______________________
```

创建JAR：

```
%cd __________
%______________________________
```

执行JAR：

```
%cd __________
%______________________________
```

作者免费招待的额外题目：这个包名称有什么问题？

there are no Dumb Questions

问： 如果用户尝试执行JAR但没有安装Java会怎样？

答： 如果没有下雨撑伞会怎样？当然不会怎样，因为没有Java虚拟机所以Java程序就不会执行。

问： 如何让用户安装Java？

答： 理想情况是自定义安装程序和应用程序一并发布。有很多厂商提供简单或高等的工具来创建installer。有些安装程序能够检测用户的计算机是否有安装合适版本的Java，如果没有的话就会帮用户安装并设定Java。Installshield、InstallAnywhere与DeployDirector等都提供安装程序工具。

有些工具还能够制作安装光盘给各种主要的Java平台使用，因此一张CD ROM就能搞定。比如在Solaris上它就会安装Solaris版的Java，在Windows上就会安装Windows版的Java。如果预算足够，你还可以来上一段耗资千万的动画片头。

要点

- 将项目组织一下以让源代码和类文件分开在不同的目录下。
- 标准的组织化结构是创建出项目目录，然后在其下建立source和classes目录。
- 将类以包来组织，并在前面加上域名称以防止命名冲突。
- 在程序源文件最前面加上包指令可以把类包进包中：

```
package com.wickedlysmart;
```

- 类必须呆在完全相对应于包结构的目录中才能包进包中。以com.wickedlysmart.Foo来说，Foo这个类必须放在com目录下wickedlysmart这个目录中。
- 要让编译过的类可以放在正确的包目录结构中，使用-d编译标识：

```
%cd source
%javac -d ../classes com/wickedlysmart/Foo.java
```

- 切换到classes目录然后指定完整的类名称来执行程序：

```
%java com.wickedlysmart.Foo
```

- 你可以把类包进JAR中，它的格式是根据pkzip制作的。
- 将描述哪个类带有main()的manifest包进JAR中可以制作出可执行的 AR文件。manifest文件是个带有像下面这样设定的文本文件，记得最后要换行才能保证正确：

```
Main-Class: com.wickedlysmart.Foo
```

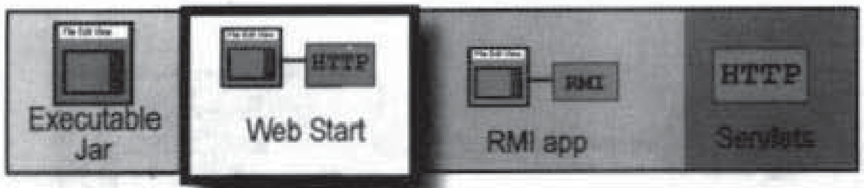
- 用下面的命令格式来创建JAR文件：

```
jar -cvfm manifest.txt MyJar.jar com
```

- JAR中的结构必须完全的符合包的目录结构。
- 以下面的命令格式来执行JAR：

```
java -jar MyJar.jar
```


可执行的JAR是很不错，但如果有办法能够制作出标准的独立的GUI在网络上发布会更好，这样就不必压缩发布CD-ROM了。如果程序还能自动地更新、替换掉有更新的部分那就更棒了，这样用户永远都会保持使用最新版……天啊！像我这种文艺美少女干嘛想这么多？



完全在本机　　介于两者之间　　完全在远程

Java Web Start

运用Java Web Start（JWS），你的应用程序可以从浏览器上执行首次启动（从web来start，懂了吗？）但它运行起来几乎像是个独立的应用程序而不受浏览器的束缚。一旦它被下载到使用者的计算机之后（首次浏览网址来下载），它就会被保存下来。

Java Web Start是个工作上如同浏览器plug-in的小Java程序（就像ActiveX组件或用浏览器打开.pdf文件出现的Acrobat Reader）。这个程序被称为Java Web Start的hepler app，主要目的是用来管理下载、更新和启动JWS程序。

当JWS下载你的程序（可执行的JAR）时，它会调用程序的main()。然后用户就可以通过JWS helper app启动应用程序而不需回到当初的网页。

这还不是最棒的，JWS还能够检测服务器上应用程序局部（例如说某个类文件）的更新——在不需要用户介入的情况下，下载与整合更新过的程序。

当然这还有点问题，比如用户要如何取得Java以及JWS。但这个问题也可以解决：从Sun下载JWS。如果装了JWS但Java版本不是最新的，Java 2 Standard Edition也会被下载到用户计算机上。

最棒的是这一切都很简单。你可以把JWS应用程序当作HTML网页或.jpg图文件一样的网络资源，设置一个链接到你的JWS应用程序上，然后就可以工作了。

反正JWS应用程序就跟从网络上下载的可执行JAR一样。

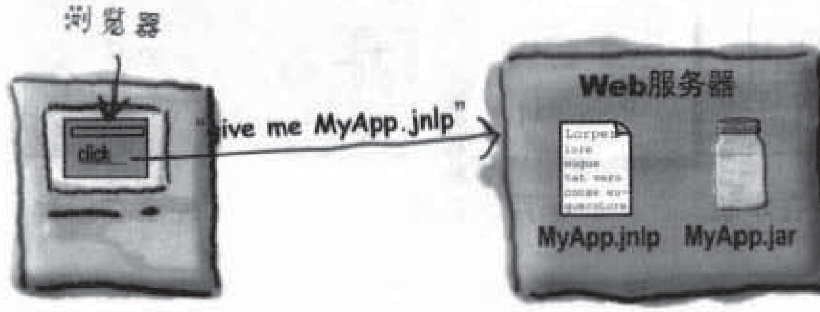
用户能通过点选网页上的某个连接来启动Java Web Start的应用程序。一旦程序下载后，它就能独立于浏览器之外来执行。事实上，Java Web Start应用程序只不过是通过网络来发布的应用程序而已。

Java Web Start的工作方式

① 客户端点击某个网页上JWS应用程序的链接（.jnlp 文件）。

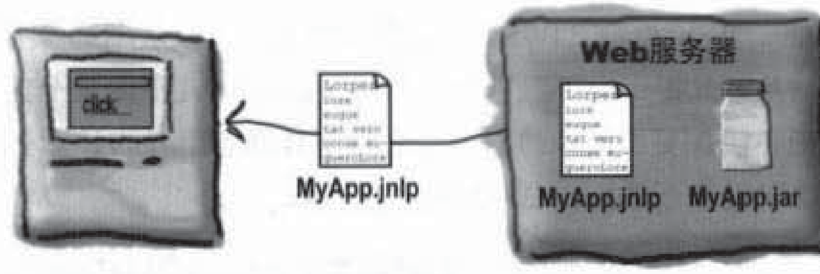
The Web page link

Click

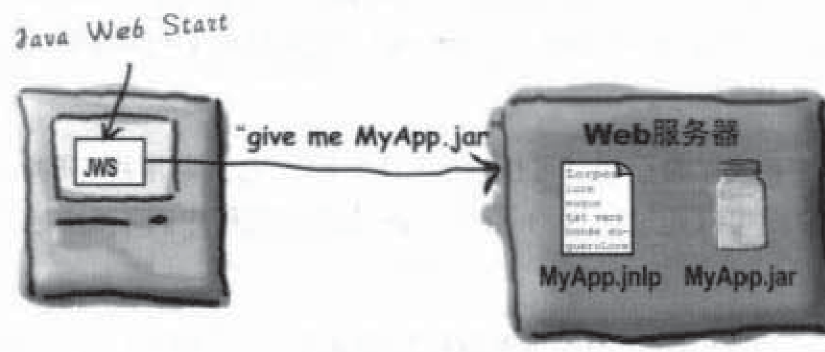


② Web服务器收到请求发出.jnlp文件（不是JAR）给客户端的浏览器。

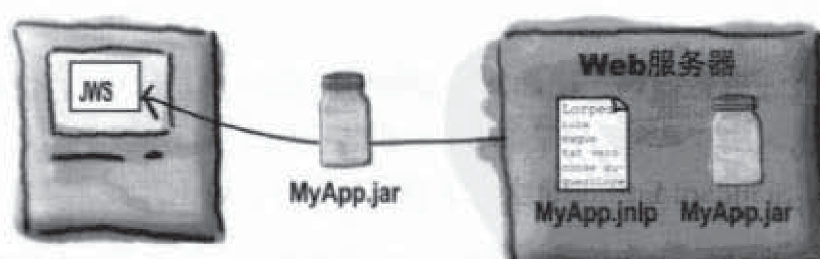
.jnlp文件是个描述应用程序可执行JAR文件的XML文件。



③ 浏览器启动Java Web Start，JWS的helper app读取.jnlp文件，然后向服务器请求MyApp.jar。

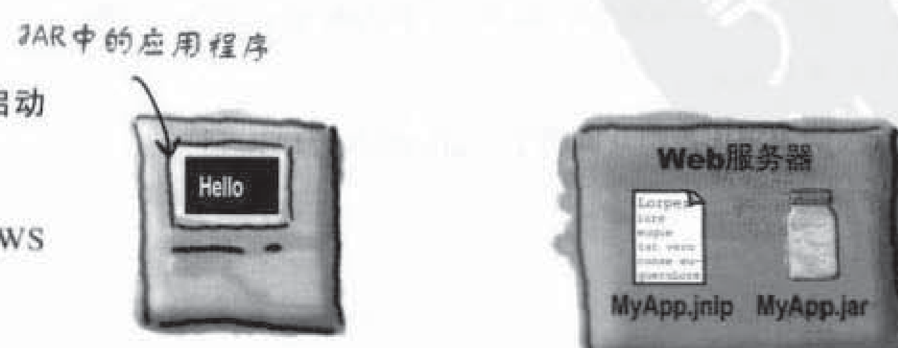


④ Web服务器发送.jar文件。



⑤ JWS取得JAR并调用指定的main()来启动应用程序。

然后用户就可以在离线的情况下通过JWS来启动应用程序。



.jnlp 文件

你需要.jnlp文件（Java Network Lanuch Protocol）来制作Java Web Start的应用程序。JWS会读取这个文件来寻找JAR并启动应用程序（调用JAR里面的main()）。.jnlp文件是个简单的XML文件，里面可以做许多设置，但至少会有下面几项：

```
<?xml version="1.0" encoding="utf-8"?>

<jnlp spec="0.2 1.0"
      codebase="http://127.0.0.1/~kathy"
      href="MyApp.jnlp">

   <information>
      <title>kathy App</title>
      <vendor>Wickedly Smart</vendor>
      <homepage href="index.html"/>
      <description>Head First WebStart demo</description>
      <icon href="kathys.gif"/>
      <offline-allowed/>
   </information>

   <resources>
      <j2se version="1.3+"/>
      <jar href="MyApp.jar"/>
   </resources>

   <application-desc main-class="HelloWebStart"/>
</jnlp>
```

codebase用来指定相关文件的起始目录，我们使用127.0.0.1来测试本机上的程序

相对于codebase的位置路径，它也可以放在某个目录下

这些tag都一定要加进去，不然程序可能会有问题。information是给helper app用的，可供显示程序的信息

设定成离线时也可执行

指定需要1.3或之后版本的Java

可执行JAR的名称

就和manifest一样描述哪个类带有main()

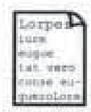
创建与部署Java Web Start的步骤

① 将程序制作成可执行的JAR



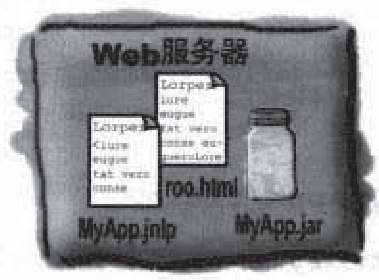
MyApp.jar

② 编写.jnlp文件



MyApp.jnlp

③ 把.jnlp与JAR文件放到Web服务器



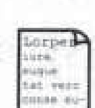
④ 对Web服务器设定新的mime类型

application/x-java-jnlp-file

这会让Web服务器以正确的header送出.jnlp数据，如此才能让浏览器知道所接收的是什么。



⑤ 设定网页连接到.jnlp文件



MyJWSApp.html

```
<HTML>
  <BODY>
    <a href="MyApp2.jnlp">Launch My Application</a>
  </BODY>
</HTML>
```

练习

先有鸡还是先有蛋？



下面有一些事件，请排列出在JWS应用程序上面发生的正确顺序。

- Web服务器送出JAR给JWS的helper app
- 浏览器启动JWS的helper app
- helper app要求取得JAR文件
- 服务器发出.jnlp给浏览器
- helper app调用JAR上的main()方法
- 用户点击网页上的链接
- 浏览器对服务器要求.jnlp文件

(1)

(2)

(3)

(4)

(5)

(6)

(7)

there are no Dumb Questions

问： Java Web Start与applet有什么不同？

答： applet无法独立于浏览器之外。applet是网页的一部分而不是单独的。浏览器会使用Java的plug-in来执行applet。applet没有类似程度的自动更新等功能，且一定得从浏览器上面执行。对JWS应用程序而言，一旦从网站上面下载后，用户不必通过浏览器就可以离线执行程序。

问： JWS 有什么安全性的限制？

答： JWS有包括用户硬盘的读写等好几项限制。但JWS自有一套API可操作特殊的对话框来打开和存储文件，因此应用程序可以在用户同意的情况下存取硬盘上特定受限区域的文件。

要点

- Java Web Start技术让你能够从网站来部署独立的客户端程序。
- Java Web Start有个必须要安装在客户端的helper app（当然也需要Java）。
- JWS程序由两个部分组成：可执行的JAR与.jnlp文件。
- .jnlp文件是用来描述JWS应用程序的XML文件。它有tag以指定JAR的名称和位置，以及带有main( )的类名称。
- 当浏览器从服务器上取得.jnlp文件时，浏览器就会启动JWS的helper app。
- JWS的helper app会读取.jnlp来判断要从服务器上下载的可执行 JAR。
- 取得JAR之后它就会调用.jnlp指定的main()。



这一章讨论包、部署和JWS。你的任务是判别下列陈述的真假。

是非题

(1) Java编译器的-d标识能让你决定.class文件要放到哪里。

(2) JAR是保存.class文件的标准目录。

(3) 创建Java Archive的时候必须要设定jar.mf文件。

(4) Java Archive里面有个辅助性的文件描述哪个类带有main()方法。

(5) Java虚拟机在使用JAR 的class之前必须先将其解压缩。

(6) Java Archive从命令行调用时必须加上-arch标识。

(7) 包结构是以层次来表示的。

(8) 对包的命名不建议使用公司的域名称。

(9) 同一源文件中的不同类可以放到不同的包中。

(10) 建议加上-p标识来编译包中的类。

(11) 编译包中的类时，完整名称必须和目录结构一致。

(12) 使用-d标识能够预防打错类名称。

(13) 将JAR解压缩时会产生meta-inf目录。

(14) 将JAR解压缩时会产生manifest.mf文件。

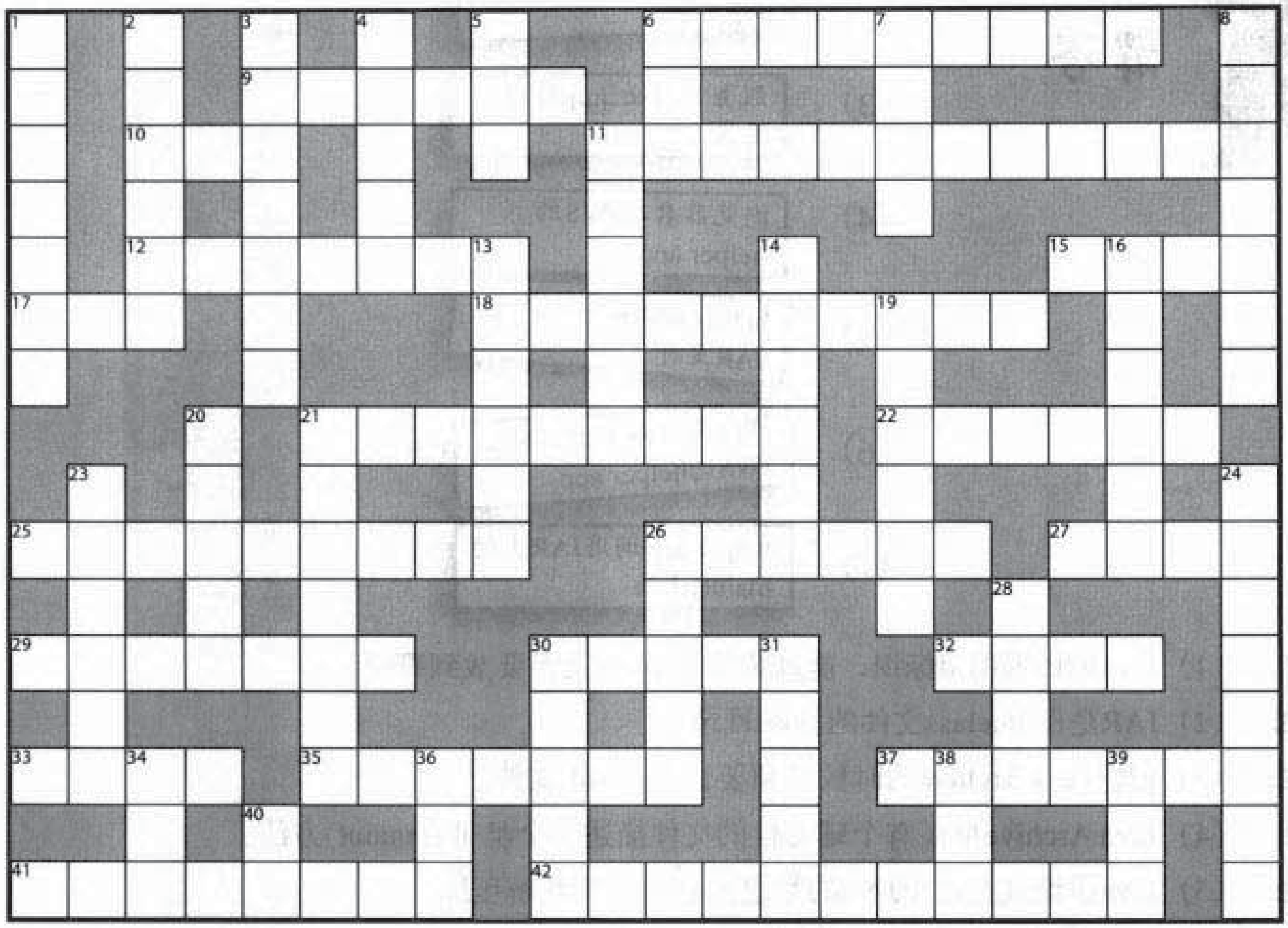
(15) JWS的helper app必须要和浏览器一起执行。

(16) JWS程序需要.html文件才能操作。

(17) JWS的main()方法是由JAR文件所指定的。

练习

字谜 7.0



书中提到的东西都有可能出现

横排提示：

6. Won't travel
9. Don't split me
10. Release-able
11. Got the key
12. I/O gang
15. Flatten
17. Encapsulated returner
18. Ship this one
21. Make it so
22. I/O sieve
25. Disk leaf
26. Mine is unique
27. GUI's target
29. Java team
30. Factory
32. For a while
33. Atomic * 8
35. Good as new
37. Pairs event
41. Where do I start
42. A little firewall

竖排提示：

1. Pushy widgets
2. ____ of my desire
3. 'Abandoned' moniker
4. A chunk
5. Math not trig
6. Be brave
7. Arrange well
8. Swing slang
11. I/O canals
13. Organized release
14. Not for an instance
16. Who's allowed
19. Efficiency expert
20. Early exit
21. Common wrapper
23. Yes or no
24. Java jackets
26. Not behavior
28. Socket's suite
30. I/O cleanup
31. Milli-nap
34. Trig method
36. Encaps method
38. JNLP format
39. VB's final
40. Java branch





(1) 用户点击网页上的链接

(2) 浏览器对服务器要求.jnlp文件

(3) 服务器发出.jnlp给浏览器

(4) 浏览器启动JWS的helper app

(5) helper app要求取得JAR文件

(6) Web服务器送出JAR给JWS的helper app

(7) helper app调用JAR上的main()方法

True (1) Java编译器的-d标识，能让你决定.class文件要放到哪里。

False (2) JAR是保存.class文件的标准目录。

False (3) 创建Java Archive的时候必须要设定jar.mf 文件。

True (4) Java Archive里面有个辅助性的文件描述哪个类带有main()方法。

False (5) Java虚拟机在使用JAR的类之前必须先将其解压缩。

False (6) Java Archive从命令行调用时必须加上-arch标识。

True (7) 包结构是以层次来表示的。

False (8) 对包的命名不建议使用公司的域名称。

False (9) 同一源文件中的不同类可以放到不同的包中。

False (10) 建议加上-p标识来编译包中的类。

True (11) 编译包中的类时，完整名称必须和目录结构一致。

True (12) 使用-d标识能够预防打错类名称。

True (13) 将JAR解压缩时会产生meta-inf目录。

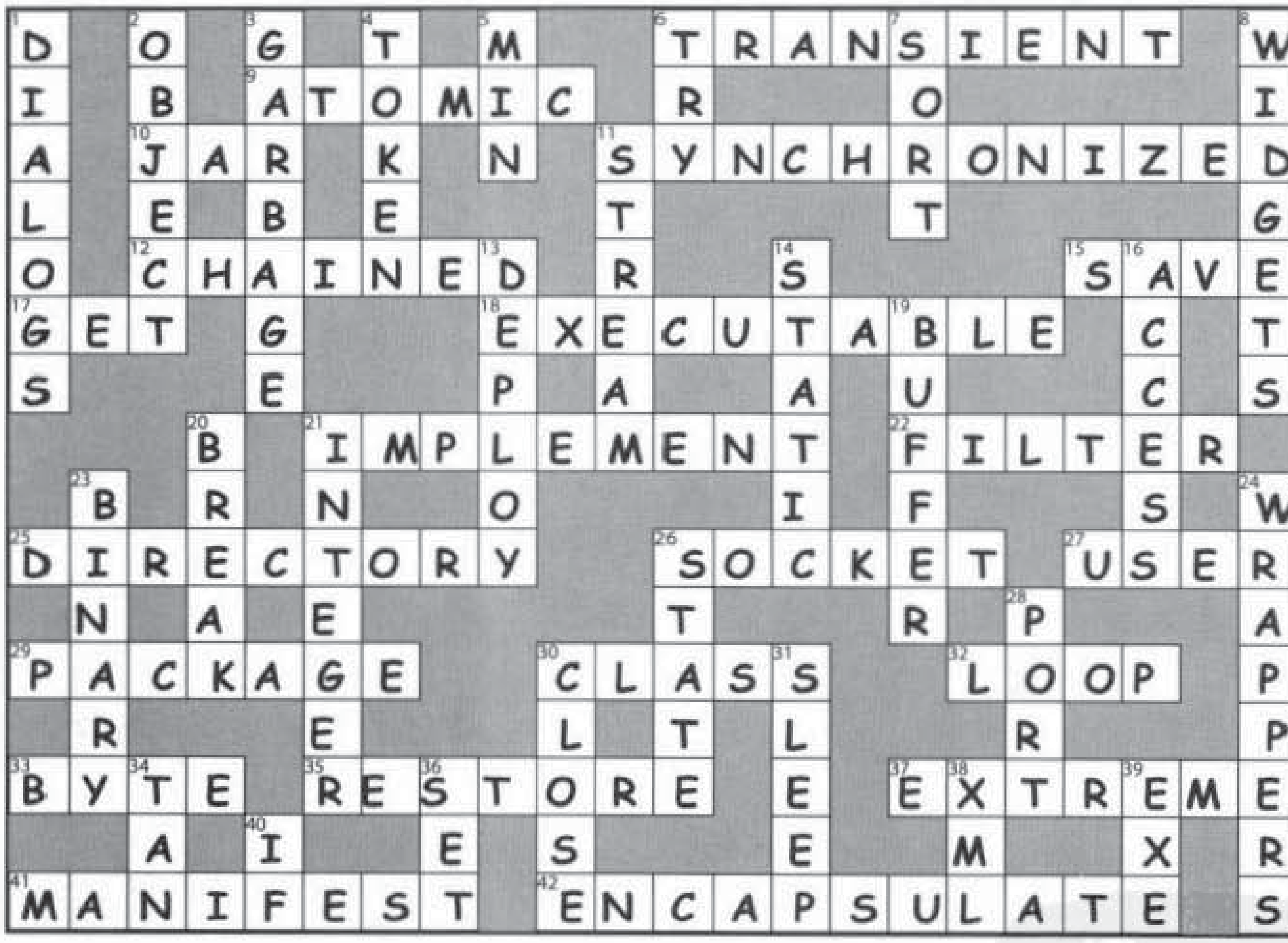
True (14) 将JAR解压缩时会产生manifest.mf文件。

False (15) JWS的helper app必须要和浏览器一起执行。

False (16) JWS程序需要.html文件才能操作。

False (17) JWS的main()方法是由JAR文件所指定的。

字谜7.0


TRANSIENT
ATOMIC
JAR
SYNCHRONIZED
CHAINED
SAVE
GET
EXECUTABLE
IMPLEMENT
FILTER
DIRECTORY
SOCKET
USER
PACKAGE
CLASS
LOOP
BYTE
RESTORE
EXTREME
MANIFEST
ENCAPSULATE

18 远程部署的RMI

分布式计算



距离不是问题。 当然啦，如果所有组件都在同一台计算机的同一个Java虚拟机的同一个堆空间上执行是最简单的，但有时候就是办不到。如果用户端只是个能够执行Java的装置怎么办？如果为了安全性的理由只能让服务器上的程序存取数据库怎么办？想象一下电子商务的情境。有时候，程序的某些部分就是得在服务器上执行，而客户端会在不同用户的计算机上执行。这一章会介绍Java的远程程序调用（Remote Method Invocation，RMI）技术。我们也会很快地看过Servlet、Enterprise Java Bean（EJB）、Jini以及EJB与Jini是如何运用RMI。最后你还会以Java创建出一个通用服务浏览器。

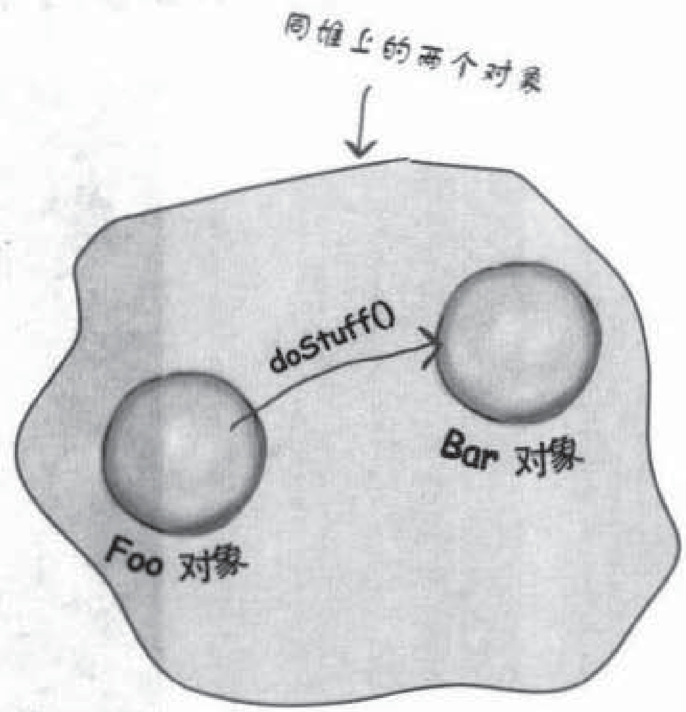


方法的调用都是发生在相同堆上的两个对象之间

本书所描述的方法调用到目前为止都是对运行在相同Java虚拟机上的对象所进行的。也就是说调用方与被调用方都是在同一个堆上。

```
class Foo {
    void go() {
        Bar b = new Bar();
        b.doStuff();
    }
    public static void main (String[] args) {
       Foo f = new Foo();
       f.go();
    }
}
```

在上面的程序中，我们知道Foo实例是被f引用，而Bar对象是被b所引用，两者都在同一个Java虚拟机上的同一个堆中。要记得Java虚拟机会负责把表示“如何取得堆上的对象”的字节组合填入引用变量中。Java虚拟机一定会知道对象，以及如何取得对象。但Java虚拟机只会知道自有堆的引用！例如你无法让Java虚拟机知道不同机器上Java虚拟机的信息。这就使Java虚拟机是不是在同一台机器上运行没有什么区别，只是使两个Java虚拟机的调用不同。



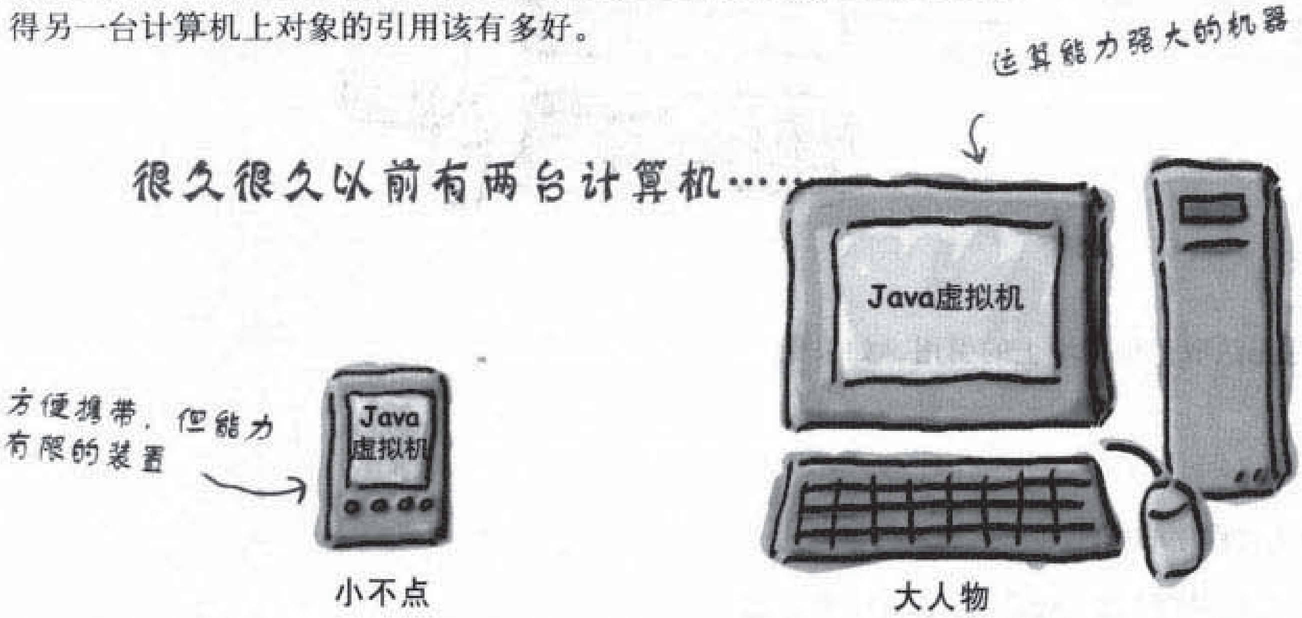
一般来说，对象的方法调用都是在相同的Java虚拟机上面进行的。

如果要调用不同机器上的对象的方法呢？

我们要知道如何从某一台计算机上面取得另一台计算机上的信息——通过Socket和输入/输出。打开另一台计算机的Socket连接，然后取得OutuputStream 来写入数据。

但如果要调用另一台计算机上，另一个Java虚拟机上面的对象的方法呢？我们当然可以自己定义和设计通信协议来调用，然后通过Socket把执行的结果再传回去。想想就知道这有多麻烦。如果能够取得另一台计算机上对象的引用该有多好。

很久很久以前有两台计算机……



大人物拥有小不点想要的东西。

就是运算能力。

小不点想要发出数据给大人物，让大人物来计算。

小不点只想要调用方法……

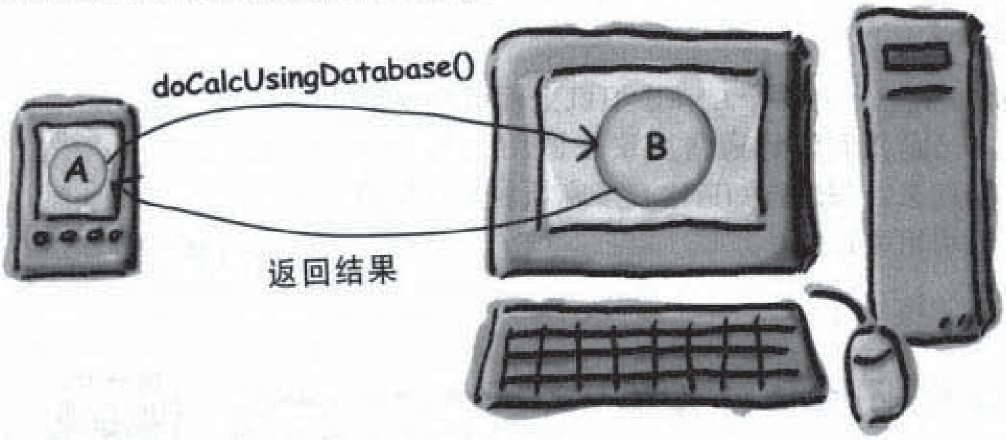
```
double doCalcUsingDatabase(CalcNumbers numbers)
```

然后取得结果。

但小不点怎样才能取得大人物的对象引用呢？

小不点的对象A想要调用大人物上对象B的方法

问题是如何取得不同机器上的对象来调用某个方法呢？



办不到

嗯……无法直接调用。你无法取得别的堆上的引用。如果像下面这样写：

Dog d = ???

无论如何d都只能引用到执行程序的堆。

如果你想要设计出某种机制能够使用Socket和输入/输出来表达你的意图（对另一台机器上面运行的对象调用方法），并且还能够像是对本机的方法调用一样，也就是说你想要调用远程的对象（像是别的堆上的），却又要像是一般的调用。

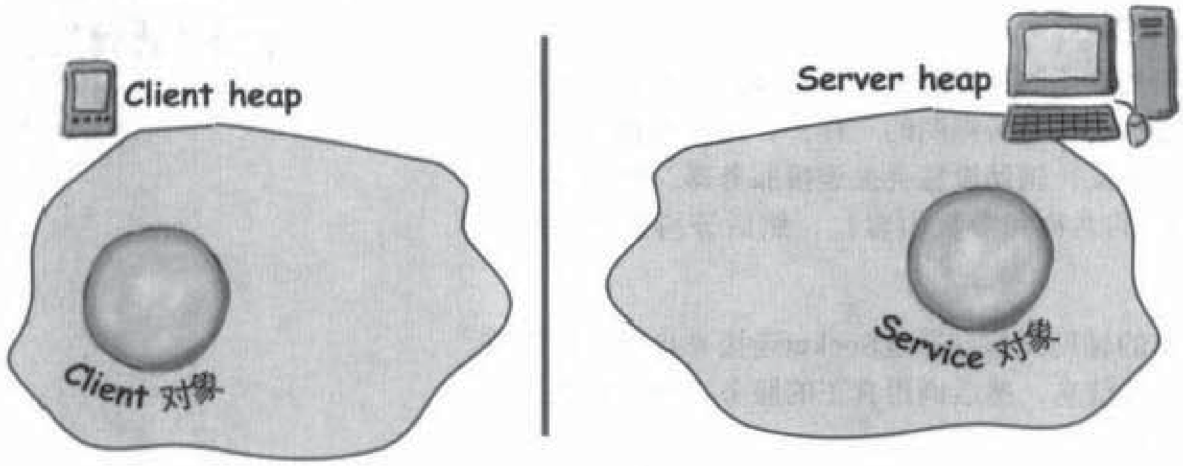
这就是RMI能够带给你的功能！

先让我们回头假设你要自己设计RMI功能。了解要如何自行创建能够帮助你学习RMI是如何工作的。

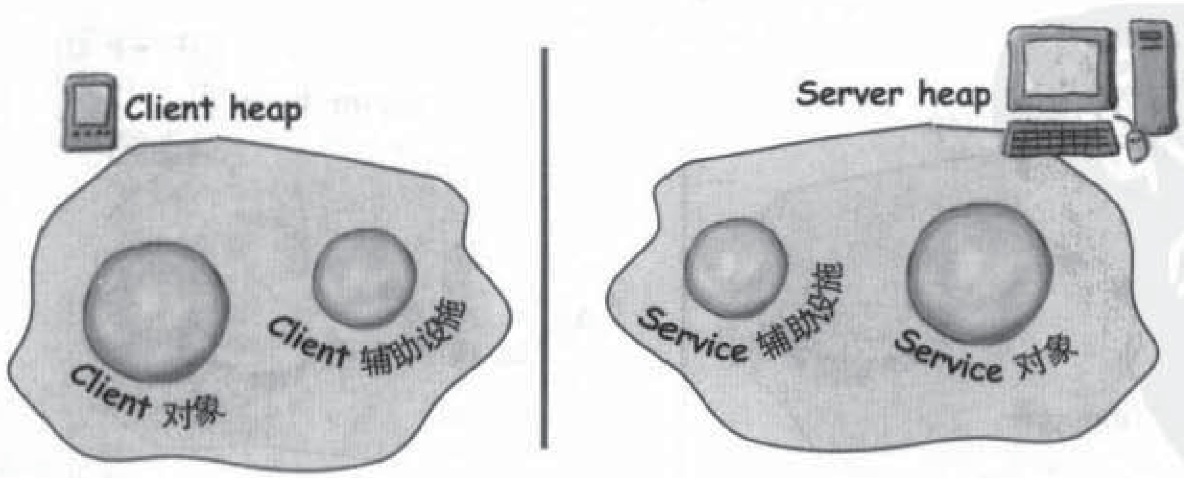
远程过程调用的设计

要创建出4种东西：服务器、客户端、服务器辅助设施和客户端辅助设施

① 创建客户端和服务器应用程序。服务器应用程序是个远程服务，是个带有客户端会调用的方法的对象。



② 创建客户端和服务器端的辅助设施（helper）。它们会处理所有客户端和服务器的低层网络输入/输出细节，让你的客户端和程序好像在处理本机调用一样。



辅助设施的任务

辅助设施是个在实际上执行通信的对象。它们会让客户端感觉上好像是在调用本机的对象。事实上正是这样。客户端调用辅助设施的方法，就好像客户端就是服务器一样。客户端是真正服务的代理（proxy）。

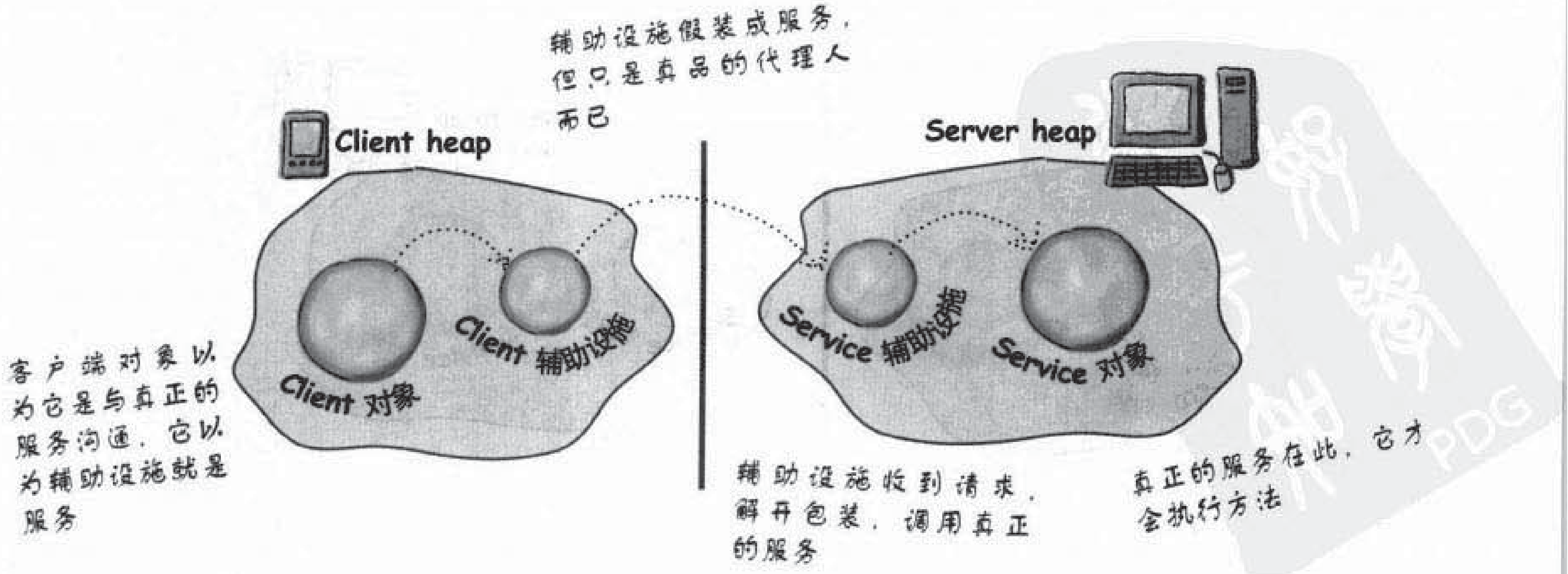
也就是说，客户端以为它调用的是远程的服务，因为辅助设施假装成该服务对象。

但客户端辅助设施并不是真正的远程服务，虽然辅助设施的举止跟它很像（因为它提供的方法跟服务所声明的一样），却没有任何真正客户端需要的方法逻辑。相反，辅助设施会去连接服务器，将调用的信息传送过去（像是方法的名称和参数内容），然后等待服务器的响应。

在服务器这端，服务器的辅助设施会通过Socket连接来自客户端设施的要求，解析打包送来的信息，然后调用真正的服务。因此对服务对象来说此调用来自于本地。

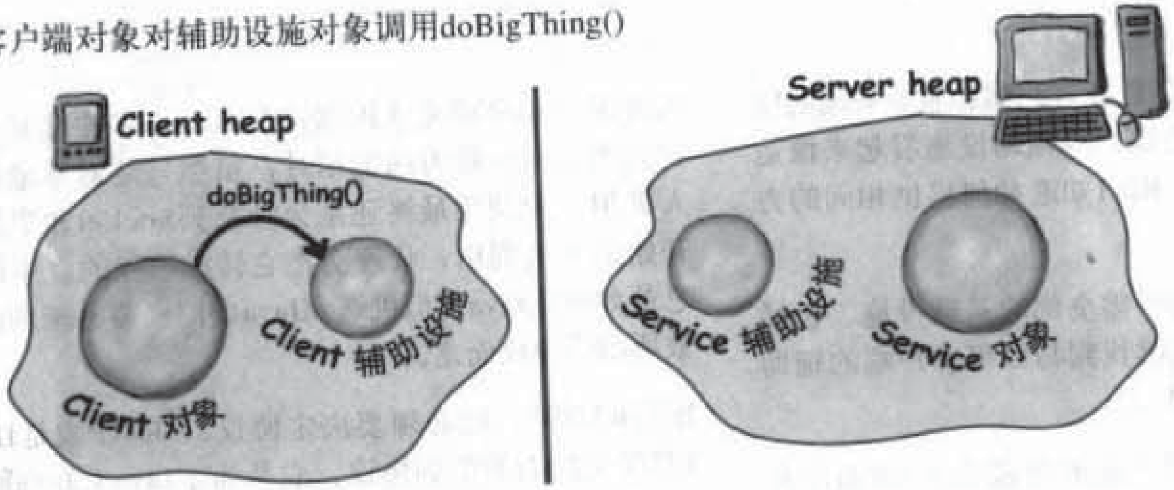
服务的辅助设施取得返回值之后就把它包装然后送回去（通过Socket的输出串流）给客户端的辅助设施。客户端的辅助设施会解开这些信息传给客户端的对象。

客户端对象看起来像是在调用远程的方法。但实际上它只是在调用本地处理Socket和串流细节的“代理”。

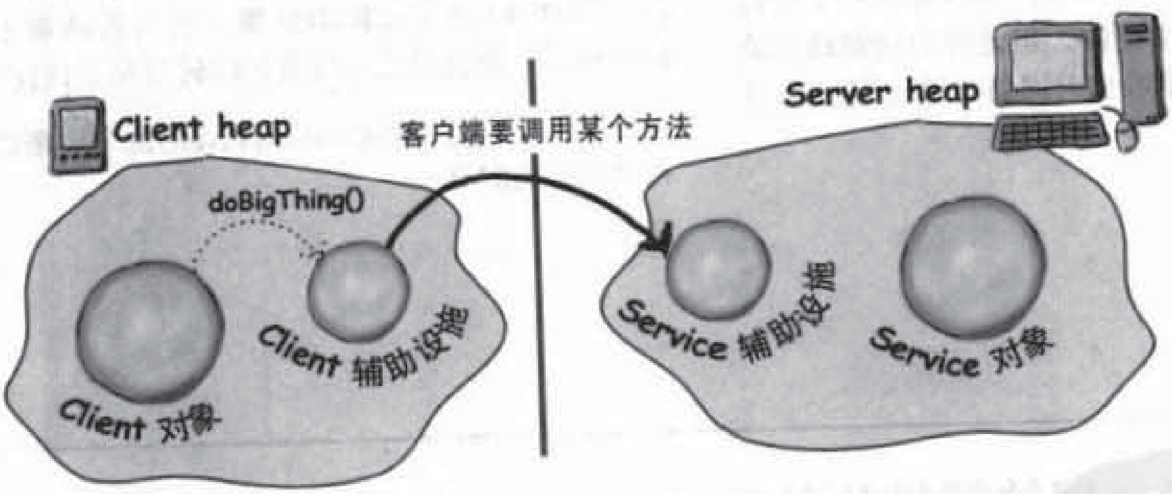


调用方法的过程

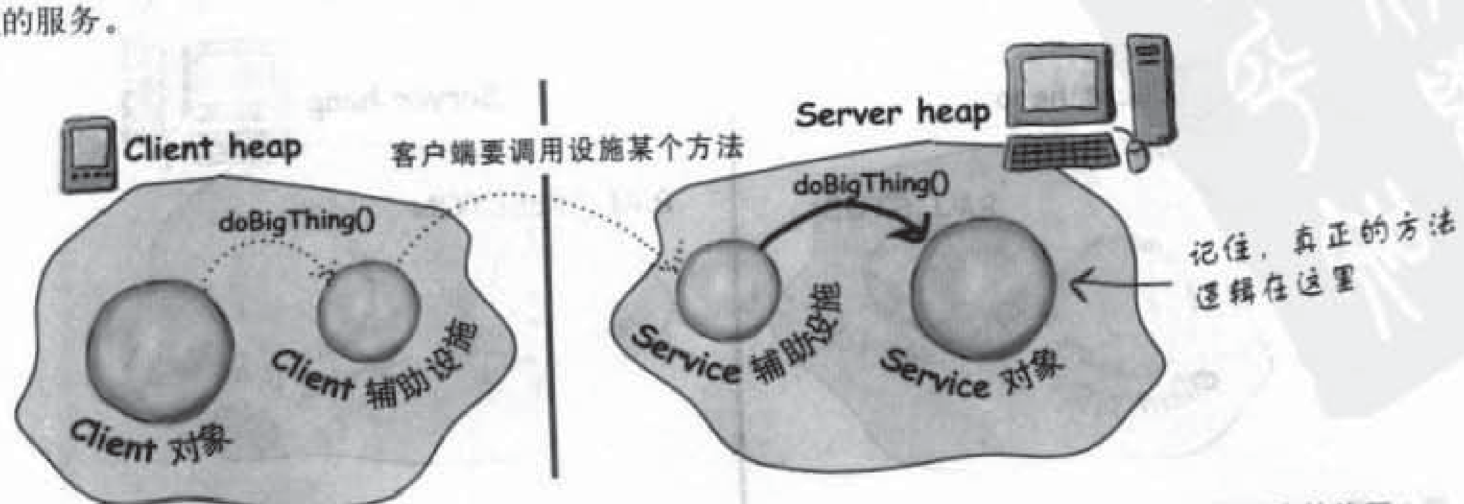
① 客户端对象对辅助设施对象调用doBigThing()



② 客户端辅助设施把调用信息打包通过网络送到服务器的辅助设施



③ 服务端的辅助设施解开来自客户端辅助设施的信息，并以此调用真正的服务。



Java RMI 提供客户端和服务器端的辅助设施对象！

在Java中，RMI已经帮你创建好客户端和服务器端的辅助设施，它也知道如何让客户端辅助设施看起来像是真正的服务。也就是说，RMI知道如何提供相同的方法给客户端调用。

此外，RMI有提供执行期所需全部的基础设施，包括服务的查询以让客户端能够找到与取得客户端的辅助设施（真正服务的代理人）。

使用RMI时，你无需编写任何网络或输入/输出的程序。客户端对远程方法的调用就跟对同一个Java虚拟机上的方法调用是一样的。

一般调用和RMI调用有一点不同。虽然对客户端来说此方法调用看起来像是本地的，但是客户端辅助设施会通过网络发出调用。那我们对网络与输入/输出有什么感觉呢?

它们都会有风险!

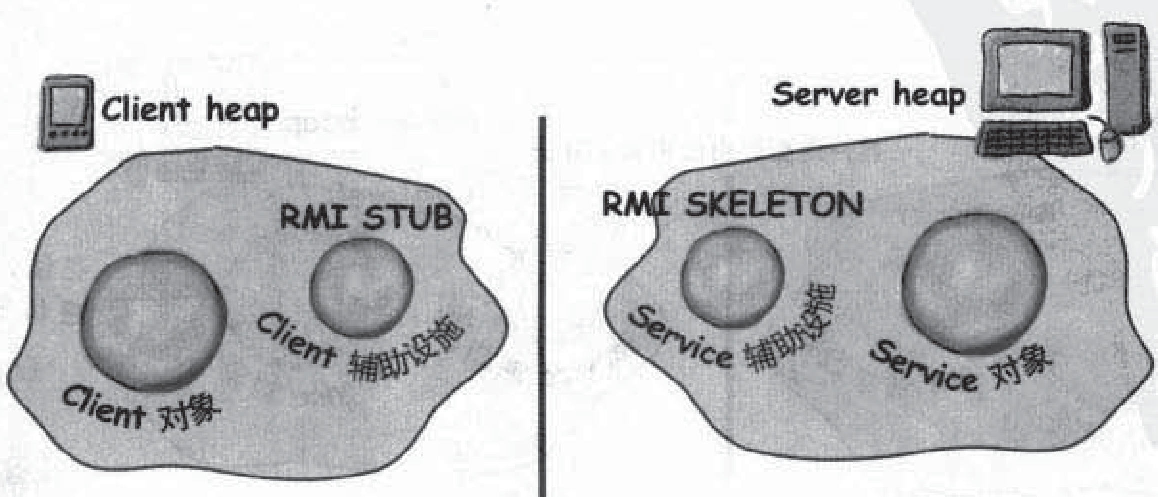
它们是会抛出异常的。

因此客户端必须要认识到这个风险。客户端必须要认知到当它对远程方法调用时，虽然只是对本地的代理人调用，此调用最终还是会涉及到Socket和串流。一开始是本机调用，代理会把它转成远程的。中间的信息是如何从Java虚拟机送到Java虚拟机要看辅助设施对象所用的协议而定。

使用RMI时，你必须要决定协议：JRMP或是IIOP。JRMP是RMI原生的协议，它是为了Java对Java间的远程调用而设计的。另外一方面，IIOP是为了CORBA（Common Object Request Broker Architecture）而产生的，它让你能够调用Java对象或其他类型的远程方法。CORBA通常比RMI麻烦，因为若两端不全都是Java 的话，就会发生一堆可怕的转译和交谈操作。

幸好，我们只需要关心Java对Java的部分，所以会使用相当简易的RMI。

在RMI中，客户端的辅助设施称为 stub，而服务器端的辅助设施称为skeleton。



创建远程服务



这是个创建远程服务的5个步骤，接下来会有每个步骤的细节。

步骤1：

创建 Remote 接口

远程的接口定义了客户端可以远程调用的方法。它是个作为服务的多态化类。stub和服务都会实现此接口！



MyRemote.java

定义客户端会调用的方法

步骤2：

实现Remote

这是真正执行的类。它实现出定义在该接口上的方法。它是客户端会调用的对象。



MyRemoteImpl.java

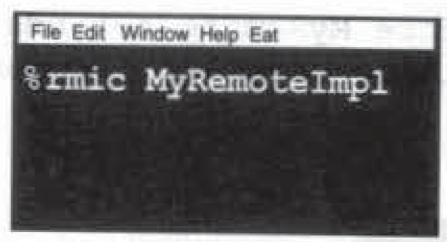
真正的服务，带有此方法的类会执行真正的工作

步骤3：

用rmic产生stub与skeleton

客户端和服务器都有 helper。你无需创建这些类或产生这些类的源代码。这都会在你执行 JDK 所附的 rmic 工具时自动地处理掉。

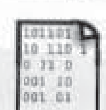
对真正实现的类执行rmic



产生出两个helper的类



MyRemoteImpl_Stub.class

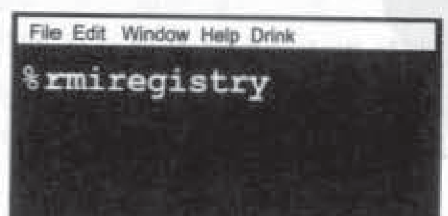


MyRemoteImpl_Skel.class

步骤4：

启动 RMI registry（rmiregistry）

rmiregistry 就像是电话簿。用户会从此处取得代理（客户端的 stub/helper 对象）。

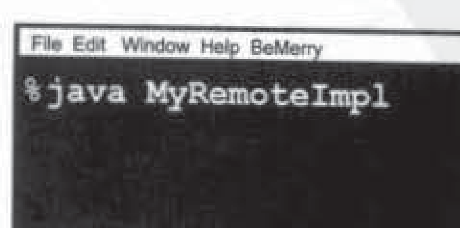


在另一个命令行上执行

步骤5：

启动远程服务

你必须让服务对象开始执行。实现服务的类会起始服务的实例并向 RMI registry 注册。要有注册后才能对用户提供服务。



步骤1：创建远程接口



MyRemote.java

① 继承java.rmi.Remote

Remote是个标记性的接口，意味着没有方法。然而它对RMI有特殊的意义，所以你必须遵守这项规则。注意这里用的是extend，接口是可以继承其他的接口。

```
public interface MyRemote extends Remote {
```

让你自己的接口声明出它和远程方法调用有关

② 声明所有的方法都会抛出RemoteException

远程的接口定义了客户端可以远程调用的方法。它是个作为服务的多态化类。也就是说，客户端会调用有实现此接口的stub，而此stub因为会执行网络和输入/输出工作，所以可能会发生各种问题。客户端必须处理或声明异常来认知这一类风险。如果方法在接口中声明异常，调用该方法的所有程序都必须处理或再声明此异常。

```
import java.rmi.*;

public interface MyRemote extends Remote {
    public String sayHello() throws RemoteException;
}
```

Remote在java.rmi中

每个远程调用都会被认为是有风险的，这样声明会强迫客户端要注意到这件事

③ 确定参数和返回值都是primitive主数据类型或Serializable

远程方法的参数和返回值必须是primitive或Serializable的。任何远程方法的参数都会被打包通过网络传送，而这是通过序列化来完成的。返回值也是一样。如果使用的是API中像是String、primitive主数据类型等主要的类型（包括数组和集合），那就没问题。如果是自定义的类型，那你就得要实现Serializable。

```
public String sayHello() throws RemoteException;
```

返回值会通过网络传送，所以必须是Serializable。参数和返回值都是这样打包传送的

步骤2：实现远程接口



MyRemoteImpl.java

① 实现Remote这个接口

你的服务必须要实现Remote——就是客户端会调用的方法。

```
public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {
    public String sayHello() {
        return "Server says, 'Hey'";
    }
    // more code in class
}
```

编译器会确保你实现出所有接口定义的方法，此例中只有一个

② 继承UnicastRemoteObject

为了要成为远程服务对象，你的对象必须要有与远程有关的功能。其中最简单的方式就是继承UnicastRemoteObject（来自java.rmi.server）以让这个父类处理这些工作。

```
public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {
```

③ 编写声明RemoteException的无参数构造函数

UnicastRemoteObject有个小问题：它的构造函数会抛出RemoteException。处理它的唯一方式就是对你的实现声明一个构造函数，如此才会有地方可以声明出RemoteException。要记得当类被初始化的时候，父类的构造函数一定会被调用，如果父类的构造函数抛出异常，你也得声明你的构造函数会抛出异常。

```
public MyRemoteImpl() throws RemoteException { }
```

真的不需写出什么，只是需要一种方式声明父类的构造函数会抛出异常

④ 向 RMI registry 注册服务

现在你已经有了远程服务，还必须要让远程用户存取。这可以通过将它初始化并加进RMI registry（它一定得要运行起来，不然此行程序就会失败）。当你注册对象时，RMI系统会把stub加到registry中，因为这是客户端所需要的。使用java.rmi.Naming的rebind( )来注册服务。

```
try {
    MyRemote service = new MyRemoteImpl();
    Naming.rebind("Remote Hello", service);
} catch(Exception ex) {...}
```

帮服务命名（客户端会靠名字查询registry），并向RMI registry注册，RMI会将stub做交换并把stub加入registry

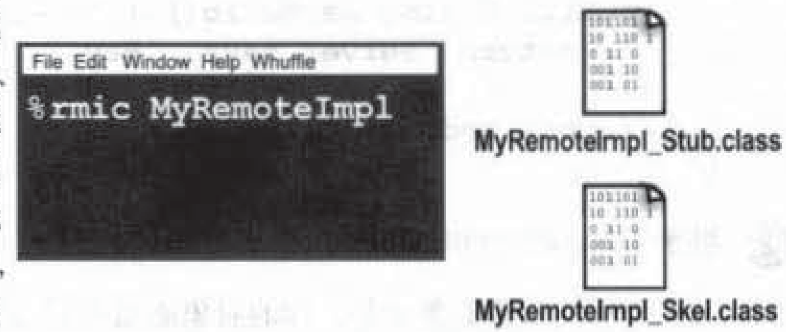
步骤3：产生stub和skeleton

① 对实现出的类（不是remote接口）执行rmic

伴随Java Software Development Kit而来的rmic工具会以服务的实现产生出两个新的类：stub和skeleton。它会按照命名规则在你的远程实现名称后面加上_Stub或_Skeleton。rmic有几个选项，包括了不产生skeleton、观察产出类的源代码或使用IIOP作为通信协议等。我们在此使用的是一般的方式。产出的类会放在当前目录下。要记住rmic必须能够找到你所实现的类，因此你或许要从实现所在的目录执行rmic（为了简化起见，我们并没有运用到包。实际上你可能需要考虑到包目录结构和完整名称）。

注意：只要类名称就行

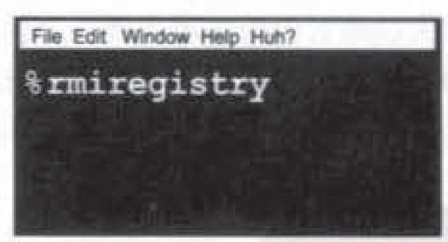
产生出两个helper对象



步骤4：执行 rmiregistry

① 调出命令行来启动 rmiregistry

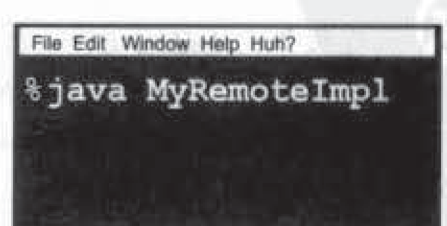
要确定你是从可以存取到该类的目录来启动。最简单的方法就是从类这个目录来运行。



步骤5：启动服务

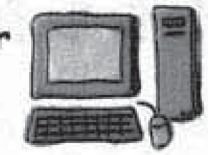
① 调用另一个命令行来启动服务

这可能是从你所实现的类上的main()或独立的启动用类来进行。在这个简单的范例中，我们把启动程序代码放在类的实现中，它的main()会将对象初始化并把它注册给RMI registry。



完整的程序代码

Server



Remote interface:

```
import java.rmi.*;

public interface MyRemote extends Remote {

    public String sayHello() throws RemoteException;
}
```

RemoteException和Remote接口都在java.rmi中

你的接口必须要继承java.rmi.Remote

所有接口中的方法都必须声明RemoteException

Remote service（实现）：

```
import java.rmi.*;
import java.rmi.server.*;

public class MyRemoteImpl extends UnicastRemoteObject implements MyRemote {

   public String sayHello() {
       return "Server says, 'Hey'";
   }

   public MyRemoteImpl() throws RemoteException { }

   public static void main (String[] args) {

       try {
           MyRemote service = new MyRemoteImpl();
           Naming.rebind("Remote Hello", service);
       } catch(Exception ex) {
           ex.printStackTrace();
       }
   }
}
```

UnicastRemoteObject是在java.rmi.server中

这是创建远程对象最简单的方式

你必须实现此接口

你得实现出接口所有的方法，但注意到你无需声明RemoteException

父类的构造函数声明了异常，所以你必须写出构造函数，因为它代表你的构造函数会调用有风险的程序代码

创建出远程对象，然后使用静态的Naming.rebind()来产生关联，所注册的名称会供客户端查询

客户端如何取得 stub 对象？

客户端必须取得stub对象，因为客户端必须要调用它的方法。这就得靠RMI registry了。客户端会像查询电话簿一样地搜索，找出上面有相符名称的服务。

必须要跟注册的名称一样

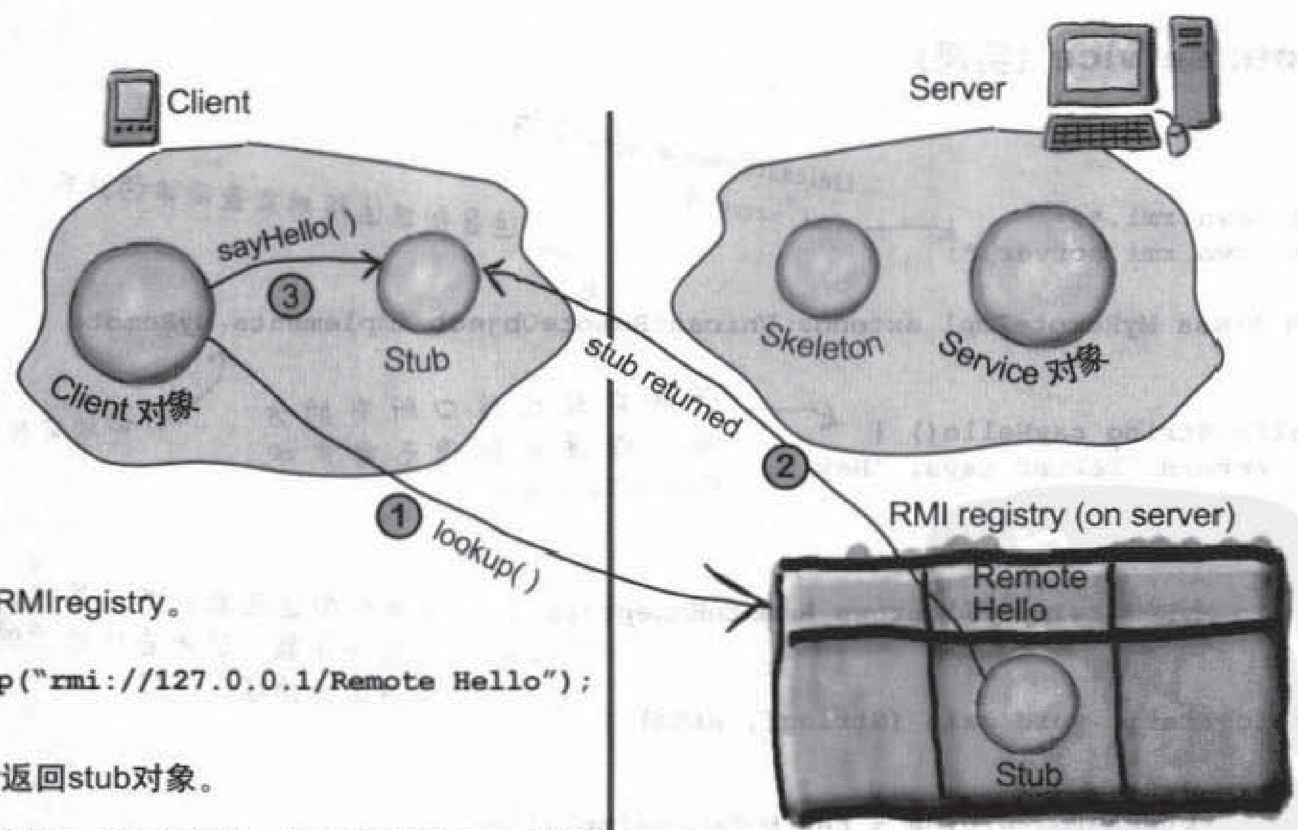
lookup()是个静态的方法

```
MyRemote service = (MyRemote)   Naming.lookup("rmi://127.0.0.1/Remote Hello");
```

客户端必须要使用与服务相同的类型，事实上客户端不需要知道服务实际上的类名称

必须要转换成接口的类型，因为查询结果会是Object类型

主机名称（host name）或IP地址



① 客户端查询 RMIregistry。

```
Naming.lookup("rmi://127.0.0.1/Remote Hello");
```

② RMI registry返回stub对象。

RMI会自动地将stub解序列化。你必须要有rmic所产生的stub类，否则客户端的stub不会被解序列化。

③ 客户端就像取用真正的服务一样的调用stub上的方法。

用户如何取得stub的类？

现在有个很有意思的问题。不管怎样做，客户端在查询服务时一定要有stub类（之前用rmic产生出来的），不然就无法在客户端解序列化。在最简单的情况下，你只要把stub的类直接交给用户就行。

但是还有更酷的方式，然而那已经超出本书的范围了。不过我们还是稍微地说明一下：最简单的方式称为“dynamic class downloading”。使用动态类下载时，stub对象会被加上注明RMI可以去哪里找到该对象的类文件的URL标记。之后在解序列化的过程中，RMI会在本机上找不到类，所以就使用HTTP的Get来从该URL取得类文件。因此你会需要一个Web服务器来提供类文件，并且也得改变客户端的某些安全性设定。这里面还有些特别的问题，不过我们就先看过动态类下载的概念就行。

完整的客户端程序代码

```
import java.rmi.*;

public class MyRemoteClient {
   public static void main (String[] args) {
        new MyRemoteClient().go();
   }

  public void go() {

     try {
        MyRemote service = (MyRemote) Naming.lookup("rmi://127.0.0.1/Remote Hello");

        String s = service.sayHello();

        System.out.println(s);
      } catch(Exception ex) {
         ex.printStackTrace();
      }
   }
}
```

Naming类是在java.rmi中

别忘了做类型转换

IP地址或host name

注册的名称

就像是调用本机的方法，但别是有风险的

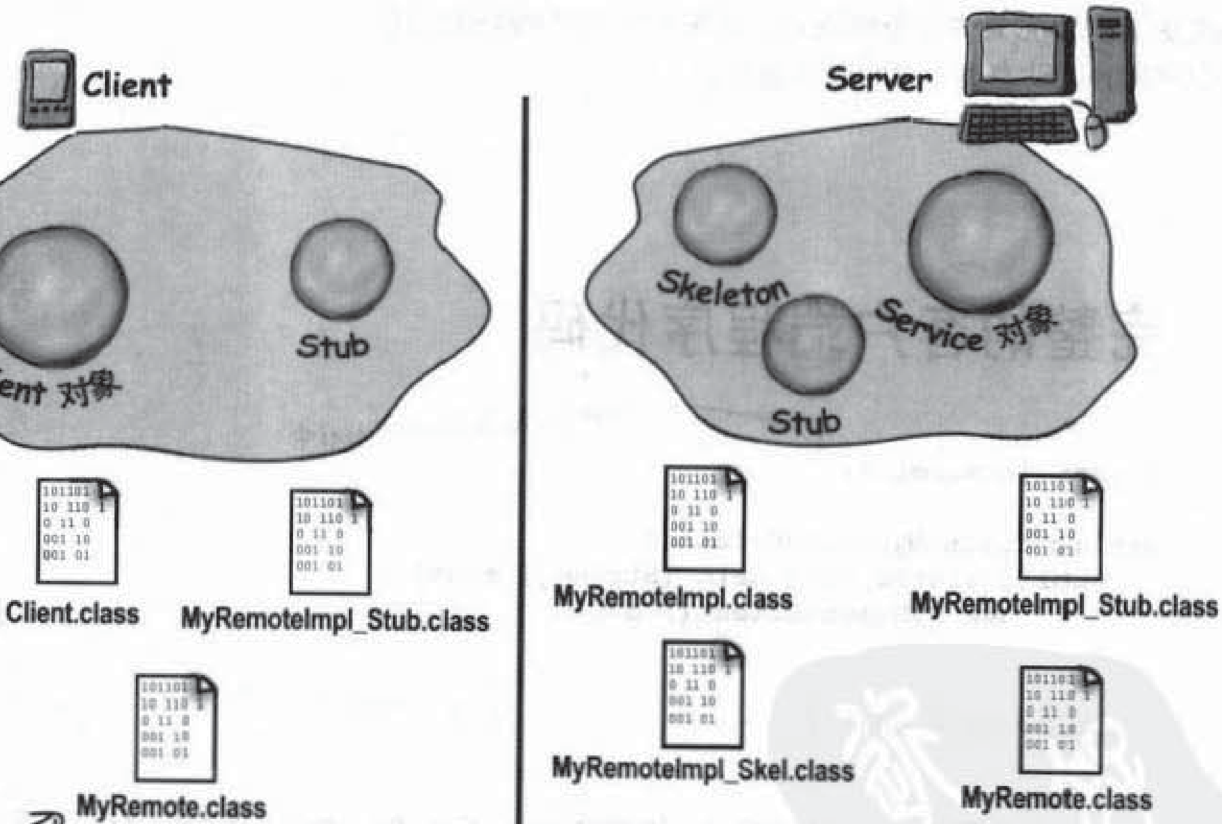
确保每台机器都有所需的类文件

使用 RMI 时程序员最常犯的3个错误：

（1）忘记在启动远程服务前启动rmiregistry（使用Naming.rebind()注册服务前rmiregistry必须启动）。

（2）忘记把参数和返回类型做成可序列化（编译器不会检测到，执行时才会发现）。

（3）忘记将stub类交给客户端。



别忘记客户端是使用接口来调用stub上的方法，客户端的Java虚拟机必须要有stub类，但客户端不会在程序代码中引用到stub类，客户端总是通过接口来操作真正的远程对象。

远程的接口

服务器上必须要有stub和skeleton，以及服务与远程的接口，它会需要stub类是因为stub会被代换成连接在RMI registry上真正的服务。

Sharpen your pencil

先有鸡还是先有蛋？



下面有一些事件，请排列出在 Java RMI 应用程序上面发生的正确顺序。

- 客户端从RMI registry取得stub
- stub将方法的调用送到服务器上
- 客户端调用stub上的方法
- 客户端查询RMI registry
- 远程服务向RMI registry做注册
- 启动RMI registry
- 远程服务被初始化

(1)

(2)

(3)

(4)

(5)

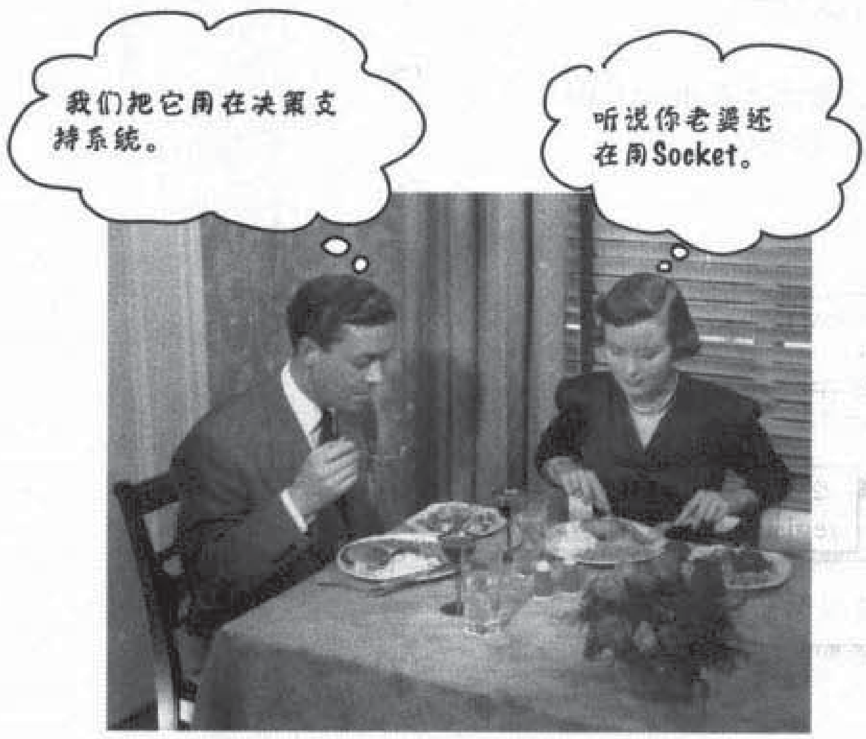
(6)

(7)

要点

- 在某堆上的对象无法进行另外堆上的对象引用。
- Java Remote Method Invocation（RMI）让你感觉上像是调用远程对象的方法，但其实不是。
- 当客户端调用远程对象的方法时，其实是调用代理上的方法，此代理被称为stub。
- stub是个处理低层网络细节的辅助性对象，它会把方法的调用包装起来送到服务器上。
- 要创建远程服务的话，你就必须要以远程接口来启动。
- 远程接口必须要extend过java.rmi.Remote这个接口，且所有的方法都必须声明RemoteException。
- 你的远程服务会实现远程接口。
- 远程服务应该要继承UnicastRemoteObject（技术上也有其他方法可以创建远程对象，但这是最简单的方式）。
- 远程服务必须要声明RemoteException的构造函数（因为父类的构造函数声明了）。
- 远程服务的对象必须要向 RMI registry 注册。
- 使用静态的Naming.rebind( )来注册远程服务。
- RMI registry必须在同一台机器上与远程服务一块执行，且必须在对象的注册之前启动。
- 客户端会以 Naming.lookup()查询远程服务。
- 几乎所有与RMI有关的都会抛出RemoteException（由编译器检查）。

是啊，有谁真的会用到RMI？











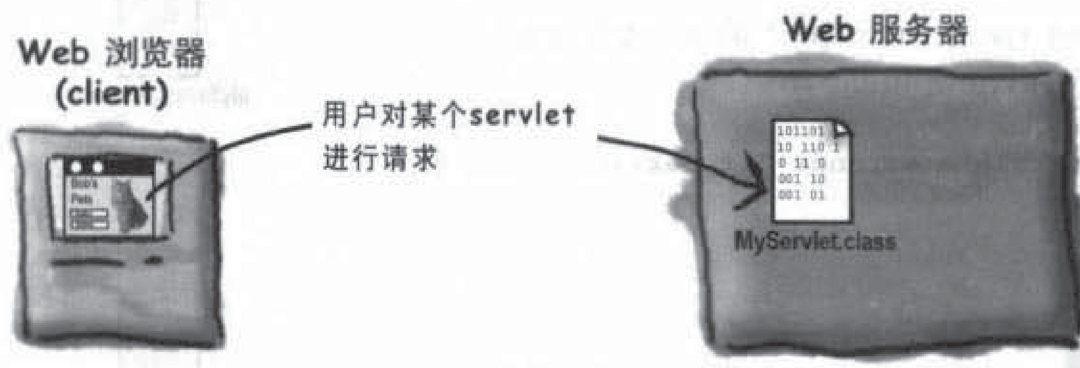
关于servlet

servlet是放在HTTP Web服务器上面运行的Java程序。当用户通过浏览器和网页交互时，请求（request）会送给网页服务器。如果请求需要Java的servlet时，服务器会执行或调用已经执行的servlet 程序代码。servlet只是在服务器上运行的程序代码，执行出用户发出请求所要的结果（例如说将数据存进数据库）。如果你本来就熟悉使用Perl来编写的CGI，那你就会知道我们说的是什么。网页开发者使用CGI或servlet来操作用户提交（submit）给服务器的信息，像是发表在讨论版上的文章。

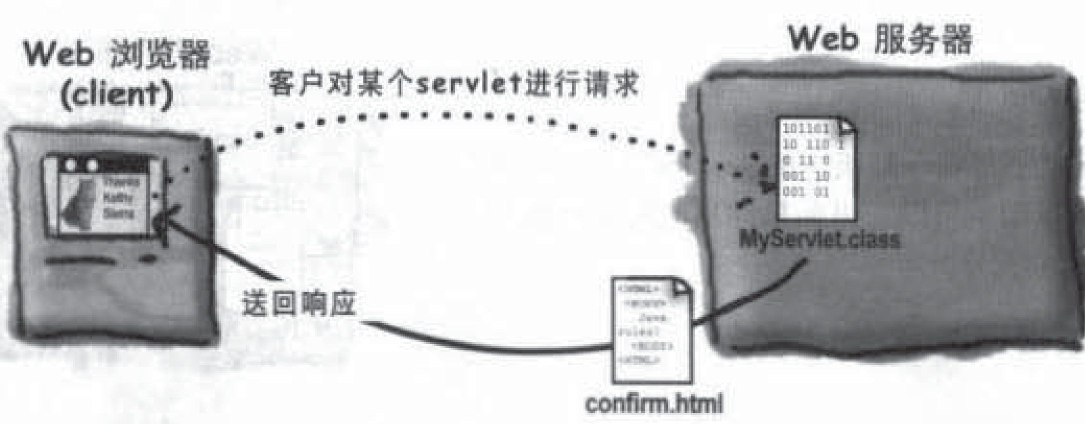
而servlet也可以使用RMI！

目前最常见的J2EE技术混合了servlet和EJB，前者是后者的用户。此时，servlet是通过RMI来与 EJB 通信的（但这与我们之前看到的程序有点不同）。

① 用户填写网页上的表格并提交。HTTP服务器收到请求，判断出是要给servlet的，就将请求传过去。

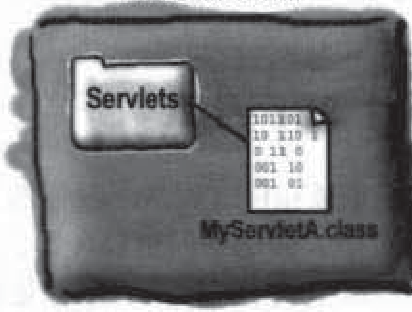


② servlet 开始执行，把数据存到数据库中，然后组合出返回给浏览器的网页。



创建并执行servlet的步骤

Web 服务器



① 找出可以存放servlet的地方

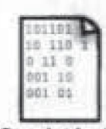
我们假设你已经有网页服务器可以运行servlet。最重要的事情是找到哪里可以存放servlet文件让服务器可以存取。如果服务器是向ISP租借的，技术支持的人应该会告诉你可以放在哪里，就像他们也会跟你说哪里可以放CGI一样。



servlets.jar

② 取得servlets.jar并添加到classpath上

servlet并不是Java标准函数库的一部分，你需要包装成servlets.jar的文件。这可从java.sun.com下载，或者从默认好可执行Java的网页服务器（例如在apache.org网站的Apache Tomcat）。没有这些类你将无法编译 servlet。

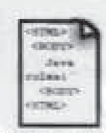


MyServletA.class

③ 通过extend过HttpServlet来编写servlet的类

servlet是个extend过HttpServlet（javax.servlet.http）的类。还有其他类型的servelet可以创建，但通常你只会使用HttpServlet。

```
public class MyServletA extends HttpServlet { ... }
```



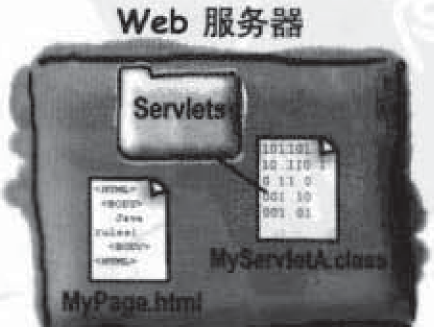
MyPage.html

④ 编写HTML来调用servlet

当用户点击引用到servlet的网页链接时，服务器会找到servlet并根据HTTP的GET、POST等命令调用适当的方法。

```
<a href="servlets/MyServletA">This is the most amazing servlet.</a>
```

Web 服务器



⑤ 给服务器设定HTML网页和servlet

这就要看你的网页服务器而定了（哪一种版本的Servlets）。ISP可能会跟你说放到Servlets目录下面就可以。但如果你用的是最新版的Tomcat，可能要多花一点时间才搞定。

很简单的servlet

除了io之外，还有两个包需要import，记住这两个是需要另行下载的，它们不是Java标准库的一部分

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
```

一般的servlet会继承HttpServlet

```
public class MyServletA extends HttpServlet {
```

Override the doGet for simple HTTP GET messages.

服务器会调用这个方法来处理请求，然后以响应对象来响应

```
    public void doGet (HttpServletRequest request, HttpServletResponse response)
                                  throws  ServletException, IOException
{
```

告诉服务器和浏览器响应结果的形式

```
        response.setContentType("text/html");
```

此对象会给我们可写回结果的输出串流

```
        PrintWriter out = response.getWriter();

        String message = "If you're reading this, it worked!";

        out.println("<HTML><BODY>");
        out.println("<H1>" + message + "</H1>");
        out.println("</BODY></HTML>");
        out.close();
    }
}
```

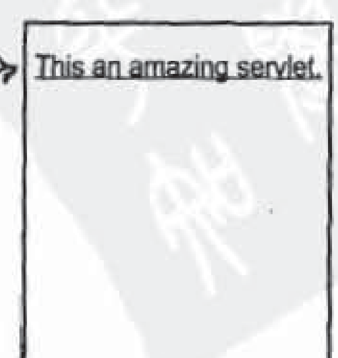
我们print的对象是个网页！就像是HTML网页一样，只是内容是由程序的输出组成的

网页看起来会像是这样：

连接到servlet的HTML网页

```
<HTML>
  <BODY>
     <a href="servlets/MyServletA">This is an amazing servlet.</a>
  </BODY>
</HTML>
```

点击链接来启动servlet



要点

- servlet是完全在 HTTP 服务器上运行的 Java 程序。
- servlet用来处理与用户交互的网页程序。例如用户提交一些信息给服务器，servlet就可以处理信息并把特定的结果以网页形式返回给用户。
- 你需要servlets.jar文件中的servelet相关包才能编译出servlet。它不是标准函数库的一部分，所以需要从java.sun.com或Web服务器供货商处取得（事实上Java 2 Enterprise Edition，也就是J2EE就带有Servlet函数库）。
- 你必须要有支持servlet的Web服务器才能运行servlet，比如apache.org的Tomcat。
- servlet必须放在特定的位置才能执行，如果Web服务器是向ISP租借的，它会告诉你应该放在哪个目录。
- 一般的servlet是继承HttpServlet并覆盖doGet()和doPost()来创建的。
- Web服务器会根据用户的请求来启动并调用servlet上对应的方法。
- servlet可以通过doGet()的响应参数取得输出串流来组成响应的网页。
- servlet要输出带有完整标识的HTML网页。

there are no Dumb Questions

问： 什么是JSP？它跟servlet有什么关系？

答： JSP代表Java Server Pages。实际上Web服务器最终会把JSP转换成servlet，但差别在于你所写出的是JSP。servlet是让你写出带有HTML输出的类，而JSP刚好相反——你会写出带有Java程序的网页！

这样就能让你在编写一般HTML网页的时候还能够同时掌握动态内容的能力，内嵌的程序代码（还有其他能够触发Java程序代码的标识）可以于执行时处理。也就是说网页内容有些部分可以是在执行时由程序制作的。

JSP的主要好处在于能更容易地编写HTM部分而不会像servlet一样出现一堆难以阅读的print命令。想象一下如果要使用多个print命令才能响应很复杂的HTML网页是多么麻烦。

但对许多应用程序来说，是无需使用到JSP的，因为servlet不需要处理动态的响应，HTML内容其实很简单。或者是因为还有一些Web服务器并不支持JSP，所以你只能使用servlet。

使用JSP的另一个好处是你可以将网页设计师与Java程序员的工作分离开来。不过这是宣传上的说法，实际上还要突破学习特定Java 标签的关卡，以为这样就能天下太平是很不切实际的想法。至少不靠工具就想办到的想法是不正确的。好消息是开发工具已经出现了，它能帮助设计师设计JSP而不用从头开始编写程序代码。

问： 花很多页讨论了RMI之后，关于servlet的讨论就这样吗？

答： 没错。RMI是Java语言的一部分，所有的RMI相关类也在标准函数库中。而servlet和JSP则不是Java语言的一部分，也不被认为是标准的扩充套件。RMI可以在任何新版的Java虚拟机上面执行，但servlet和JSP需要正确设定好的Web服务器和servlet的容器。这就是为什么说这个部分已经超出本书的范围。但你还是可以从《Head First Servlets & JSP》这本书寻求更多帮助。

闲着没事，来把专家术语学习机改造成servlet

我们无法抗拒把第1章的专家术语学习机改成网络化的想法。servlet也是程序，Java程序可以调用其他程序，因此servlet能够调用学习机的方法。只要把学习机的类放到 servlet 的目录就行（学习机的程序代码在下一页）。




```
import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class KathyServlet extends HttpServlet {
    public void doGet (HttpServletRequest request, HttpServletResponse response)
                                          throws ServletException, IOException  {

        String title = "PhraseOMatic has generated the following phrase.";

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        out.println("<HTML><HEAD><TITLE>");
        out.println("PhraseOmatic");
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + title + "</H1>");
        out.println("<P>" + PhraseOMatic.makePhrase());
        out.println("<P><a href=\"KathyServlet\">make another phrase</a></p>");
        out.println("</BODY></HTML>");

        out.close();
    }
}
```

看到没：servlet可以调用其他类的方法

适用于servlet的专家术语学习机

这跟第1章的版本稍有不同。原来是在 main( ) 中运行整个程序，且必须在命令行上重新执行程序才能收到新的组合。新版本会在调用静态的 makePharse()方法时返回 String。如此一来，你可以在任何程序中来调用此方法取得随机组合的字词。

注意：不要键入String[]中的破折号！实际上的程序并没有在两个双引号之间换行。

```
public class PhraseOMatic {
   public static String makePhrase() {

     // make three sets of words to choose from
     String[] wordListOne = {"24/7","multi-Tier","30,000 foot","B-to-B","win-win","front-end", "web-
based","pervasive", "smart", "six-sigma","critical-path", "dynamic"};

     String[] wordListTwo = {"empowered", "sticky", "valued-added", "oriented", "centric", "distrib-
uted", "clustered", "branded","outside-the-box", "positioned", "networked", "focused", "leveraged",
"aligned", "targeted", "shared", "cooperative", "accelerated"};

     String[] wordListThree = {"process", "tipping point", "solution", "architecture", "core compe-
tency", "strategy", "mindshare", "portal", "space", "vision", "paradigm", "mission"};

    // find out how many words are in each list
    int oneLength = wordListOne.length;
    int twoLength = wordListTwo.length;
    int threeLength = wordListThree.length;

    // generate three random numbers, to pull random words from each list
    int rand1 = (int) (Math.random() * oneLength);
    int rand2 = (int) (Math.random() * twoLength);
    int rand3 = (int) (Math.random() * threeLength);

    // now build a phrase
    String phrase = wordListOne[rand1] + " " + wordListTwo[rand2] + " " + wordListThree[rand3];

    // now return it
    return ("What we need is a " + phrase);
  }
}
```

Enterprise JavaBeans：打了类固醇的RMI

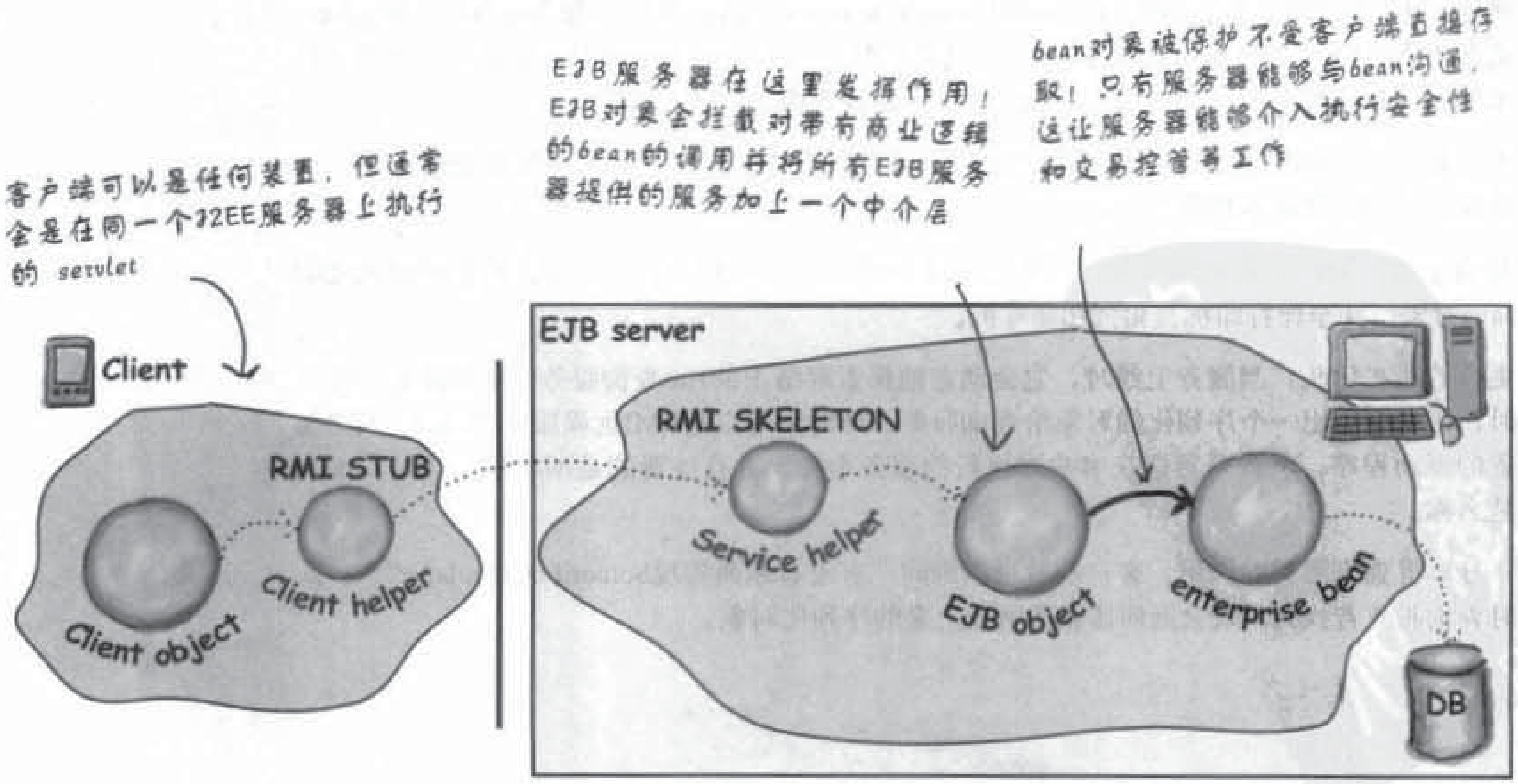
RMI很适合编写并运行远程服务。但你不会单独使用RMI来执行Amazon或eBay网站服务。对大型的企业级应用程序来说，你需要更多更好的功能。你需要交易管理、大量并发处理（全世界的人一起上来抢购“哈利·波特——消失的蜜饯”）、安全性和数据库管理等。为此，你会需要enterprise application server。

用Java术语来说，这就是Java 2 Enterprise Edition（J2EE）服务器。J2EE服务器包括了Web服务器和Enterprise JavaBeans（EJB）服务器。就跟servlet一样，EJB已经超过了本书的范围，且EJB也无法用简短的范例展示，但我们会对它的工作方式稍加说明。

（译注：业界已经出现一股反对EJB的声音，理由就跟你不会开十八轮大货车去市场买菜一样，工具和技术并不是越大越重就越合适。）

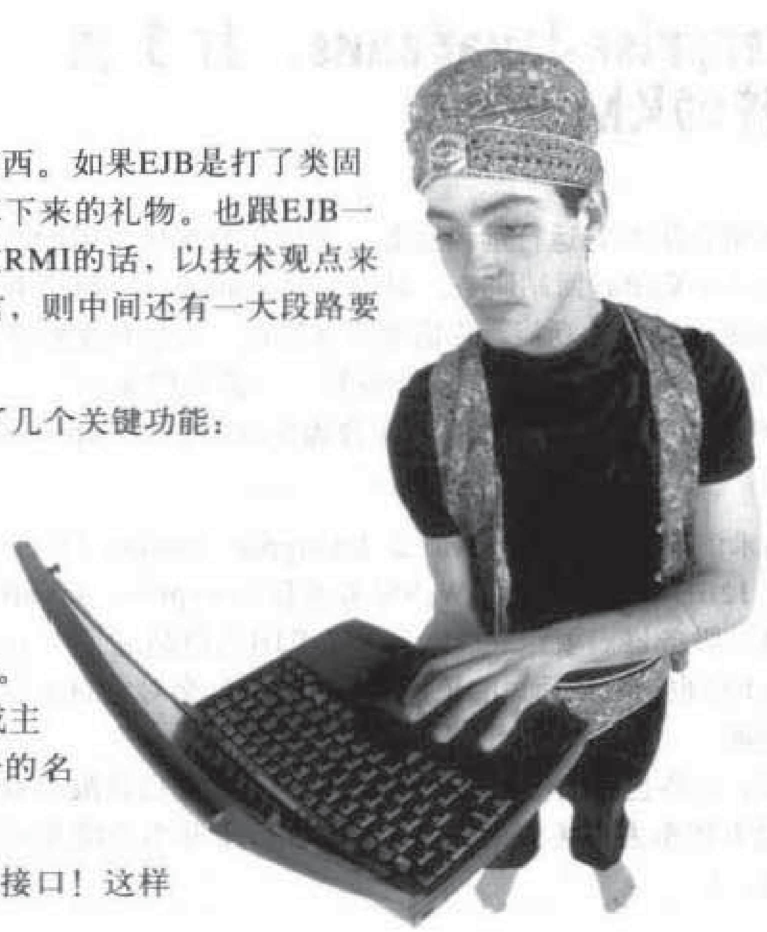
EJB服务器具有一组你光靠RMI不会有的服务，比如交易管理、安全性、并发性、数据库和网络功能等。

EJB服务器作用于RMI调用和服务层之间。



这只是完整EJB架构的一部分

最后，来点Jini魔法



我们是超爱Jini的。我们认为Jini应该是Java最棒的东西。如果EJB是打了类固醇的RMI，Jini就是长出翅膀的RMI。真是个天上掉下来的礼物。也跟EJB一样，我们无法在这本书讨论Jini的细节，但若你搞定RMI的话，以技术观点来说那就已经接近一半真相的一半。以观念的理解而言，则中间还有一大段路要走。

Jini也是使用 RMI（虽然也可以用别的协议），但多了几个关键功能：

（1）adaptive discovery（**自适应探索**）。

（2）self-healing networks（**自恢复网络**）。

要知道RMI的客户端得先取得远程服务的地址和名称。客户端的查询程序代码就要带有远程服务的IP地址或主机名（因为RMI registry就在上面）以及服务所注册的名称。

但使用Jini时，用户只需知道一件事：服务所实现的接口！这样就行。

哪怎么找到的呢？秘诀就在于Jini的lookup service。Jini的查询服务比RMI registry更强更有适应性。因为Jini会在网络上自动的广告。当查询服务上线时，它会使用IP组播技术送出信息给整个网络说："大爷我在这里，想找东西就问我"。

不只是这样。如果客户端在查询服务已经广播之后上线，客户端也可以发出信息给整个网络说："那个谁在不在啊？"。

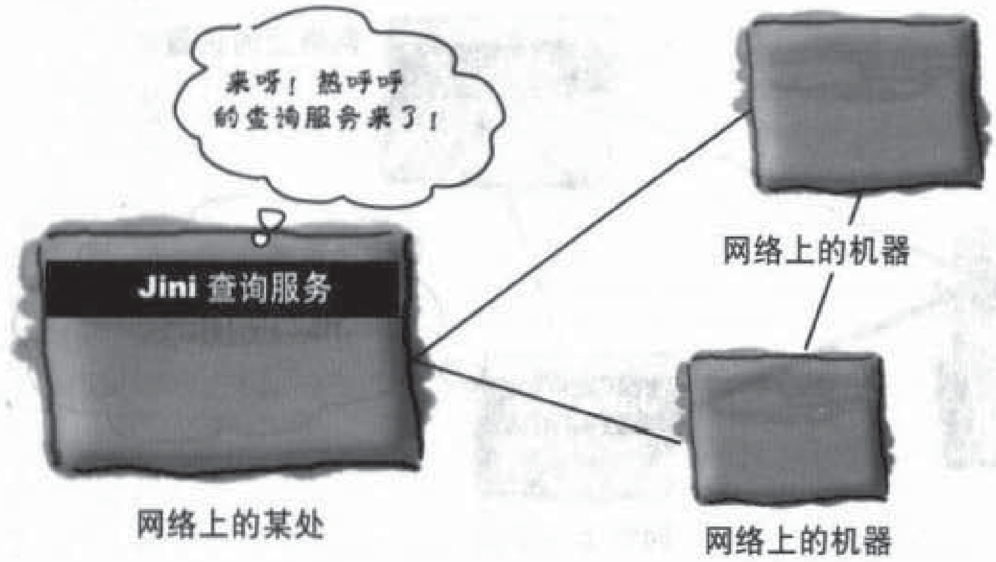
其实你感兴趣的不是查询服务，而是已经注册的服务。像RMI远程服务、其他可序列化的Java对象，甚至像打印机、相机和咖啡机。

更棒的还在后头：当服务上线时，它会动态地探索网络上的Jini查询服务并申请注册。注册时，服务会送出一个序列化的对象给查询服务。此对象可以是RMI远程服务的stub、网络装置的驱动程序，甚或是可以在客户端执行的服务本身，并且注册的是所实现的接口，而不是名称。

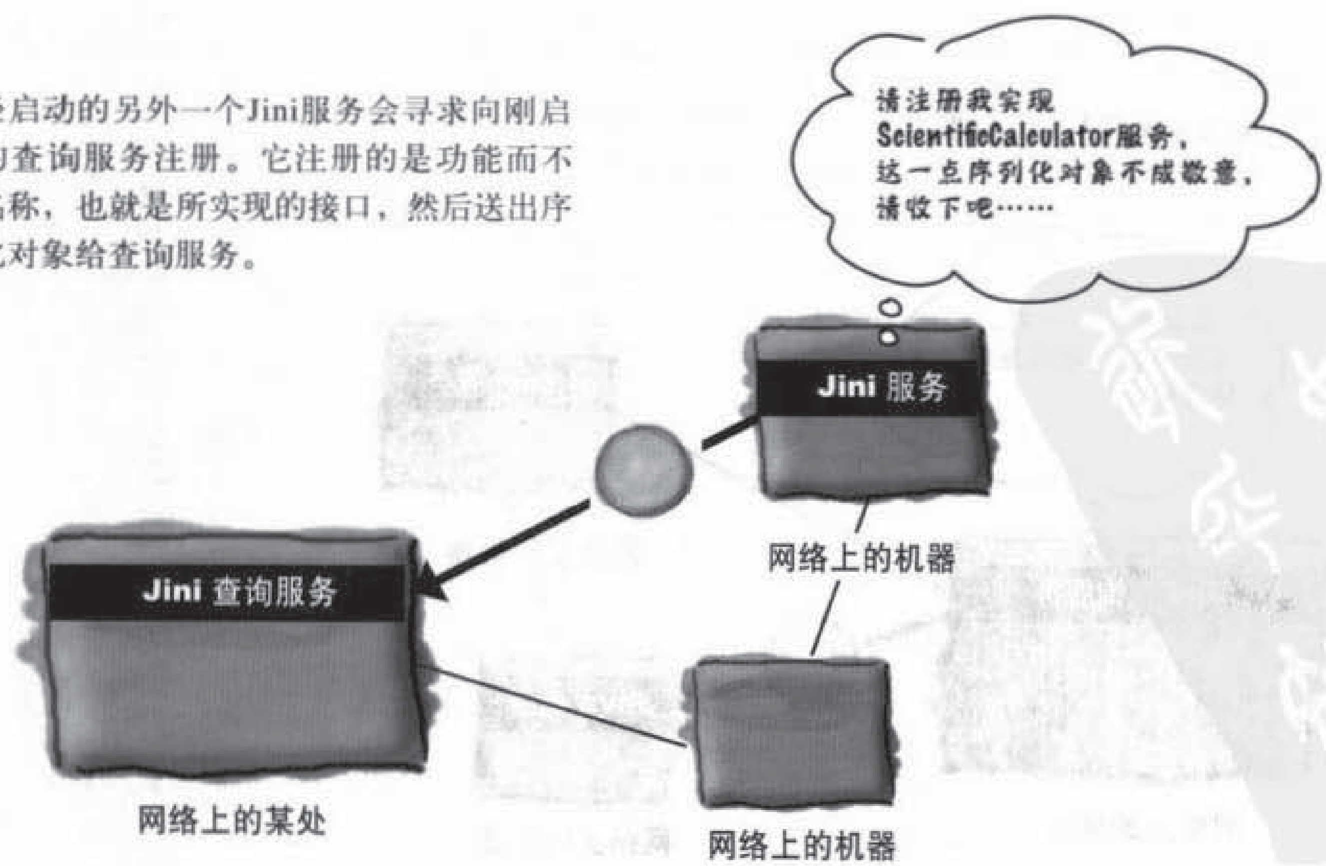
一旦取得查询服务的引用，客户端就可以询问"有没有东西实现ScientificCalculator?"。此时查询服务若找到，就会返回该服务所放上来的序列化对象。

自适应探索的运作

① Jini查询服务在网络上启动，并使用IP组播技术为自己做宣传。

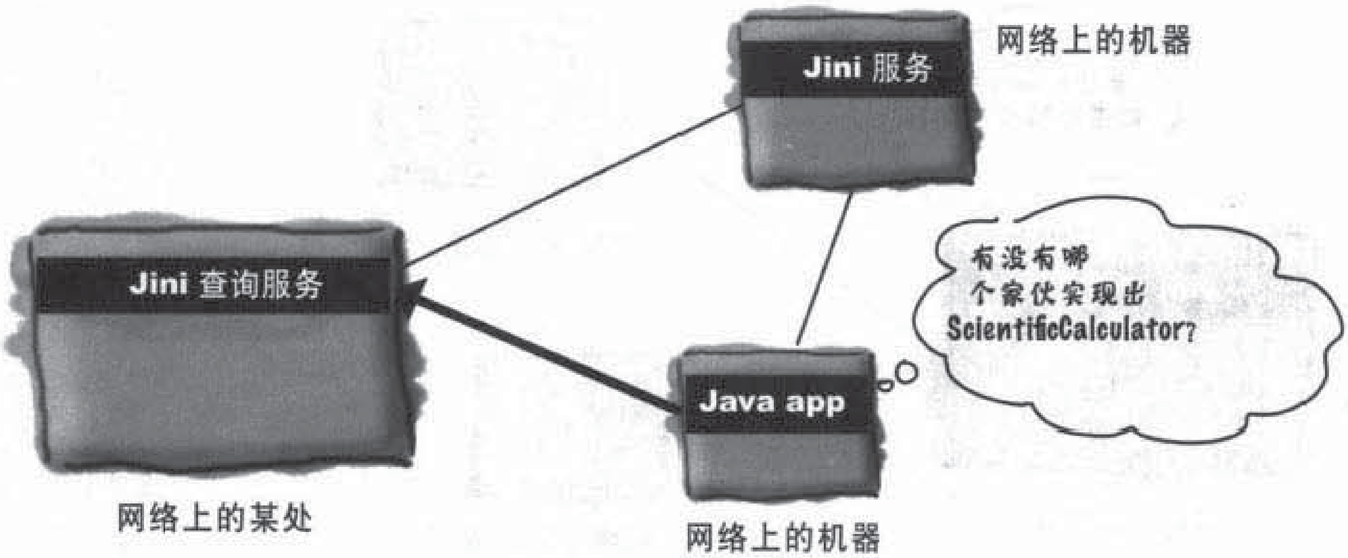


② 已经启动的另外一个Jini服务会寻求向刚启动的查询服务注册。它注册的是功能而不是名称，也就是所实现的接口，然后送出序列化对象给查询服务。

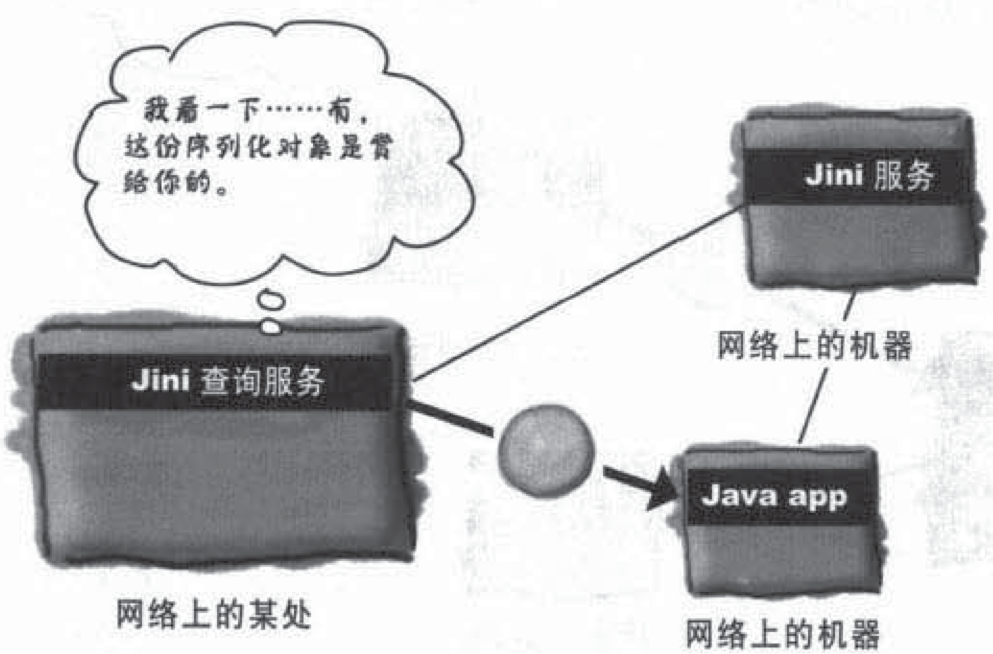


自适应探索的运作（续）

③ 网络客户想要取得实现ScientificCalculator的东西，可是不知道哪里有，所以就问查询服务。

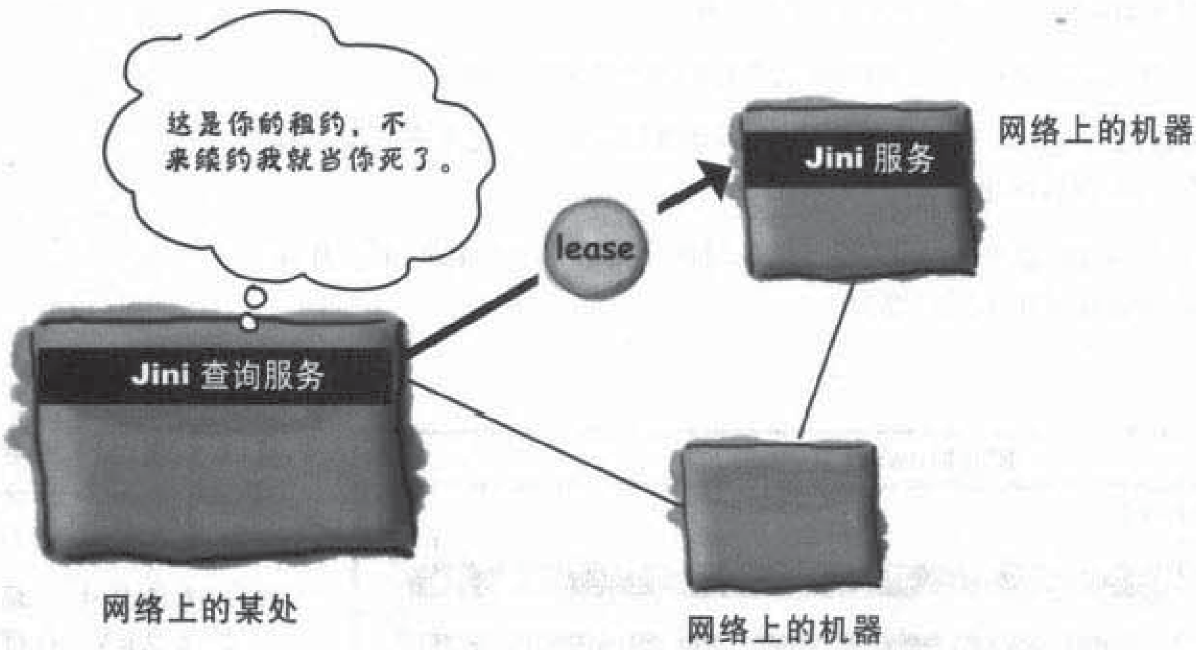


④ 查询服务响应查询的结果。

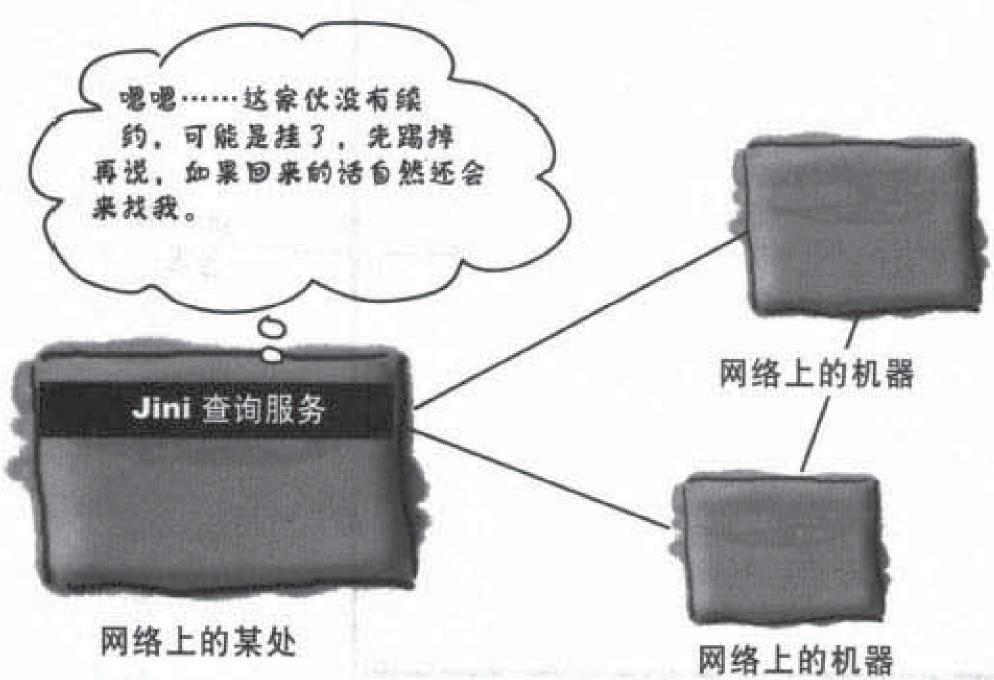


自恢复网络的运作

① 某个Jini服务要求注册，查询服务回给一份租约。新注册的服务必须要定期更新租约，不然查询服务会假设此服务已经离线了。查询服务会力求呈现精确完整的可用服务网络状态。



② 因为关机所以服务离线，因此也没有更新租约，查询服务就把它踢掉。



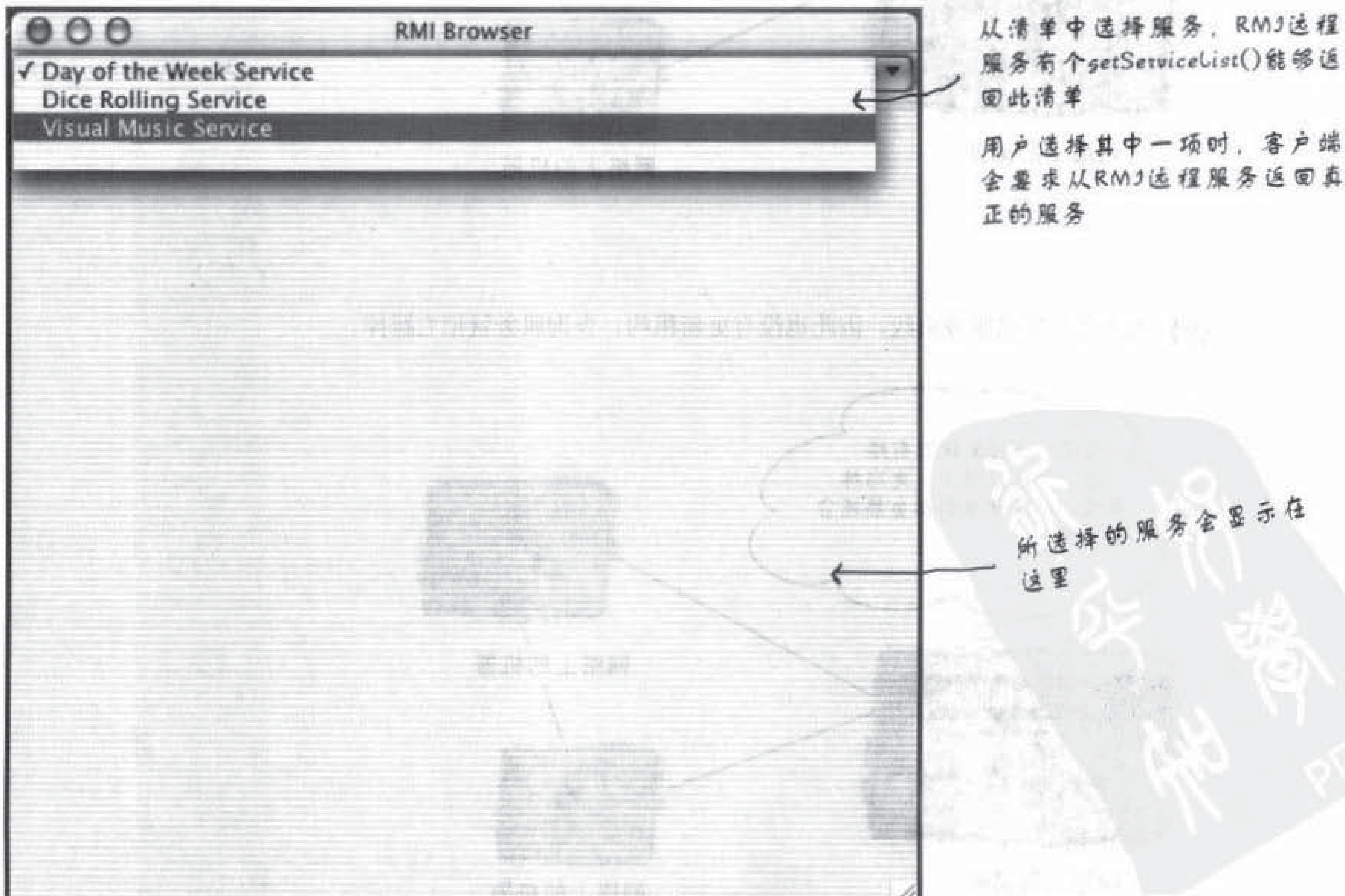
终极任务：通用服务浏览器

我们要做一个没有Jini功能的程序，但很容易实现。它会让你体验Jini，却只有用到RMI。事实上我们的程序与Jini应用程序的主要差别只在于服务是如何探索的。相对于Jini的查询服务会自动地对所处的网络做广告，我们使用的是必须与远程服务在同一台机器上执行的RMI registry，这当然不会自动的声明。

且服务也不会自动地向查询服务做注册，我们必须将它注册给RMI registry。

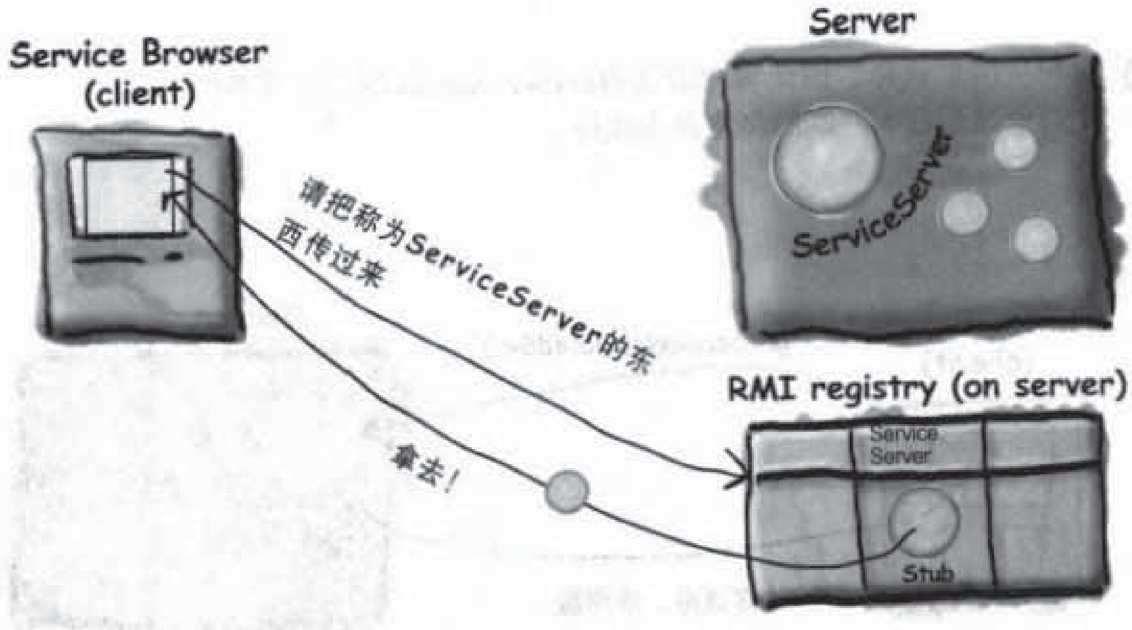
一旦用户在RMI registry找到服务，应用程序其余的部分就跟Jini的方式几乎一模一样（当然还少了租约和续约机制这回事）。

通用服务浏览器就像是个特殊化的网页浏览器，只是所展示的并非HTML网页。此服务浏览器会下载并显示出交互的Java图形界面。

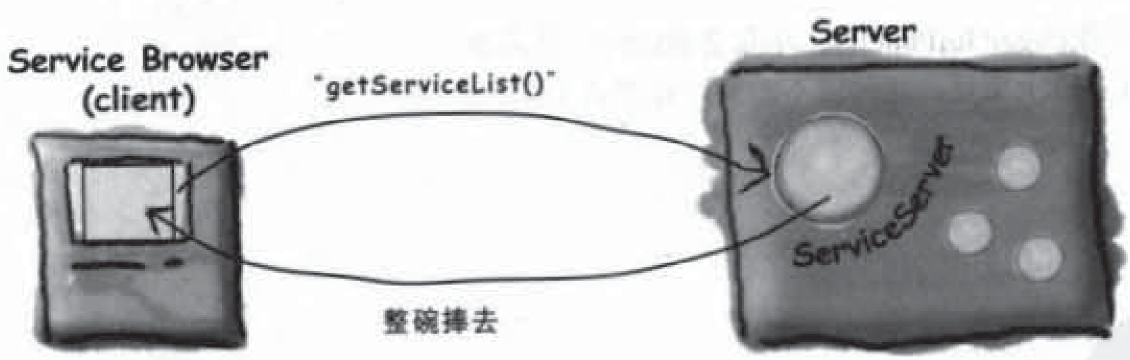


它的运作方式：

① 用户启动并查询在RMI registry上注册为ServiceServer的服务，然后取回stub。



② 客户端对stub调用getServiceList()。它会返回服务的数组。

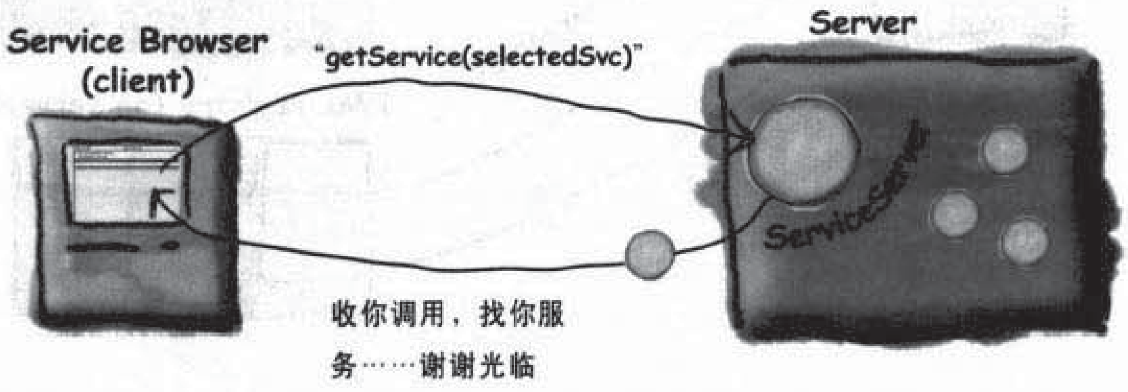


③ 客户端以GUI显示出服务对象的清单。



运作方式（续）：

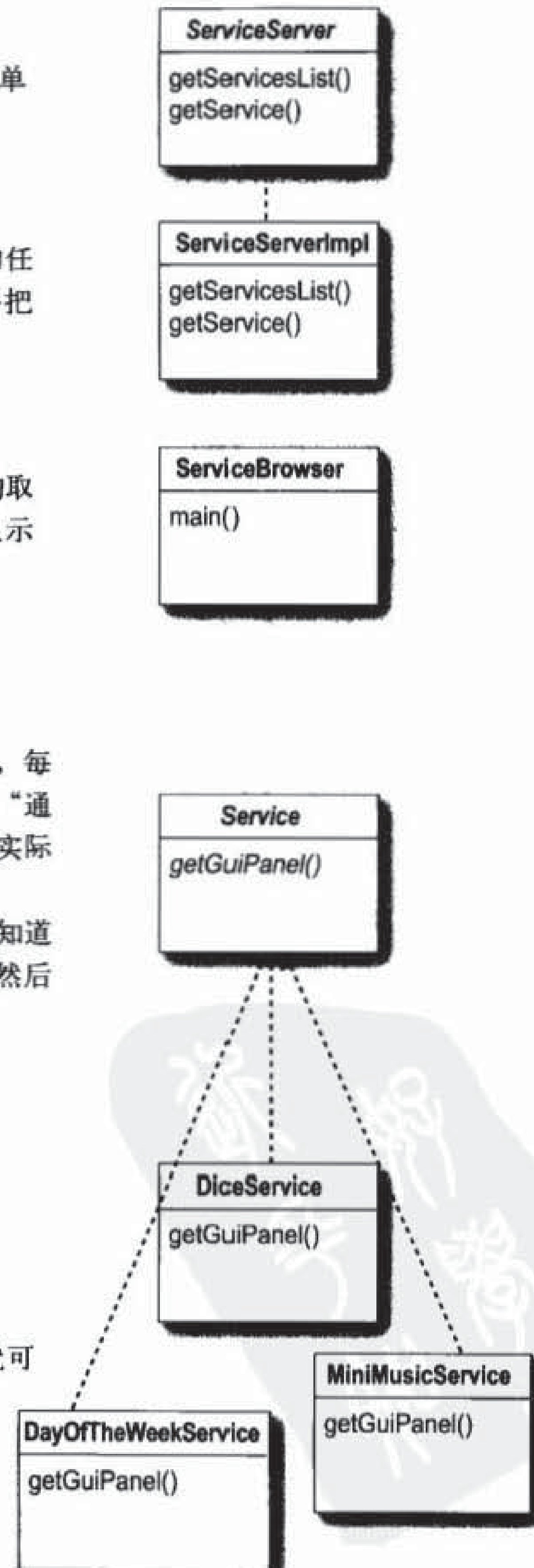
④ 用户从清单选择，因此客户端调用getService()取得实际服务的序列化对象然后在客户端的浏览器上执行。



⑤ 客户端调用刚取得序列化对象的getGuiPanel()。此服务的GUI会显示在浏览器中，且用户可与它在本机上交互。此时就不需要远程服务了。



类与接口



- **ServiceServer这个接口实现了Remote。**
  那是给远程服务用的RMI普通接口（让远程服务写出取得服务清单的 方法）。

- **ServiceServerImpl这个类实现了ServiceServer。**
  这是实际的RMI远程服务（有继承UnicastRemoteObject），它的任务是初始化并存储全部的服务（会被传送给客户端的东西），并把自己登记给RMI registry。

- **ServiceBrowser。**
  这是客户端的类，它创建出很简单的GUI，在RMI registry中查询取得ServiceServer的stub，然后调用它的远程服务取得服务清单并显示在GUI上面。

- **Service。**
  它是最关键的部分。这个简单的接口只有一个getGuiPanel()方法，每个要传送给客户端的服务都得实现这个接口。这样才会让浏览器“通用”！实现这个接口之后，服务才能克服客户端完全不知道服务实际类的问题。至少来的服务会有getGuiPanel()方法。
  客户端调用getService(selectedSvc)后取得序列化对象，然后在不知道是什么类的情况下调用一定会有的getGuiPanel()来取得JPanel，然后放到浏览器上开始与用户交互。

- **DiceService实现Service。**
  需要骰子吗？没问题，取用这个服务就会出现虚拟的骰子。

- **实现Service的MiniMusicService。**
  还记得我们之前做过音乐录像带吗？我们把它改装成服务，你就可以不停地播放听了想打人的音乐。

- **DayOfTheWeekService实现Service。**
  输入你的生日就可以知道今天是星期几。

interface ServiceServer (远程接口)

```
import java.rmi.*;
public interface ServiceServer extends Remote {
    Object[] getServiceList() throws RemoteException;
    Service getService(Object serviceKey) throws RemoteException;
}
```

一般的RMI接口，定义出两个远程服务要实现的方法

interface Service (GUI 服务要实现的部分)

```
import javax.swing.*;
import java.io.*;

public interface Service extends Serializable {
    public JPanel getGuiPanel();
}
```

定义任何通用服务都得要实现的setGuiPanel()这个方法，因为继承Serializable，所以能够自动地序列化，而这也是又通过网络传送服务所必须的机制

class ServiceServerImpl (远程的实现)

```
import java.rmi.*;
import java.util.*;
import java.rmi.server.*;
```

一般的RMI实现

```
public class ServiceServerImpl extends UnicastRemoteObject implements ServiceServer  {

    HashMap serviceList;
```

服务会被存储在HashMap集合中

```
    public ServiceServerImpl() throws RemoteException {
       setUpServices();
    }

    private void setUpServices() {
       serviceList = new HashMap();
       serviceList.put("Dice Rolling Service", new DiceService());
       serviceList.put("Day of the Week Service", new DayOfTheWeekService());
       serviceList.put("Visual Music Service", new MiniMusicService());
    }
```

构造函数被调用时会将实际的通用服务初始化

创建服务并将String名称放进HashMap中

```
    public Object[] getServiceList() {
       System.out.println("in remote");
       return serviceList.keySet().toArray();

    }
```

客户端会调用它以取得服务的清单，我们会送出Object的数组，只带有HashMap的key，实际服务会到用户要求时才通过getService()送出

```
    public Service getService(Object serviceKey) throws RemoteException {
       Service theService = (Service) serviceList.get(serviceKey);
       return theService;
    }
```

这个方法会通过key名称来返回HashMap中相对应的服务

```
    public static void main (String[] args) {
       try {
         Naming.rebind("ServiceServer", new ServiceServerImpl());
       } catch(Exception ex) {
           ex.printStackTrace();
      }
       System.out.println("Remote service is running");
    }
}
```

PDG

class ServiceBrowser (用户端)

```
import java.awt.*;
import javax.swing.*;
import java.rmi.*;
import java.awt.event.*;


public class ServiceBrowser {

   JPanel mainPanel;
   JComboBox serviceList;
   ServiceServer server;


   public void buildGUI() {
      JFrame frame = new JFrame("RMI Browser");
      mainPanel = new JPanel();
      frame.getContentPane().add(BorderLayout.CENTER, mainPanel);

      Object[] services = getServicesList();   ← 此方法执行RMI registry查询，取得stub并
                                                  调用getServiceList()

                                                  把此服务添加到JComboBox里面，JComboBox知道
      serviceList = new JComboBox(services);   ← 如何显示数组中的字符串


      frame.getContentPane().add(BorderLayout.NORTH, serviceList);

      serviceList.addActionListener(new MyListListener());

      frame.setSize(500,500);
      frame.setVisible(true);

  }

   void loadService(Object serviceSelection) {
       try {
          Service svc = server.getService(serviceSelection);

          mainPanel.removeAll();                 把实际的服务添加到GUI上的mainPanel中
          mainPanel.add(svc.getGuiPanel());
          mainPanel.validate();
          mainPanel.repaint();
        } catch(Exception ex) {
         ex.printStackTrace();
        }
   }
```

```
    Object[] getServicesList() {
       Object obj = null;
       Object[] services = null;

       try {

          obj = Naming.lookup("rmi://127.0.0.1/ServiceServer");

       }
       catch(Exception ex) {
        ex.printStackTrace();
       }
       server = (ServiceServer) obj;


       try {

         services = server.getServiceList();

       } catch(Exception ex) {
          ex.printStackTrace();
       }
       return services;

    }

    class MyListListener implements ActionListener {
       public void actionPerformed(ActionEvent ev) {

            Object selection =  serviceList.getSelectedItem();
            loadService(selection);
       }
     }

    public static void main(String[] args) {
       new ServiceBrowser().buildGUI();
    }
}
```

执行RMI查询，取得stub

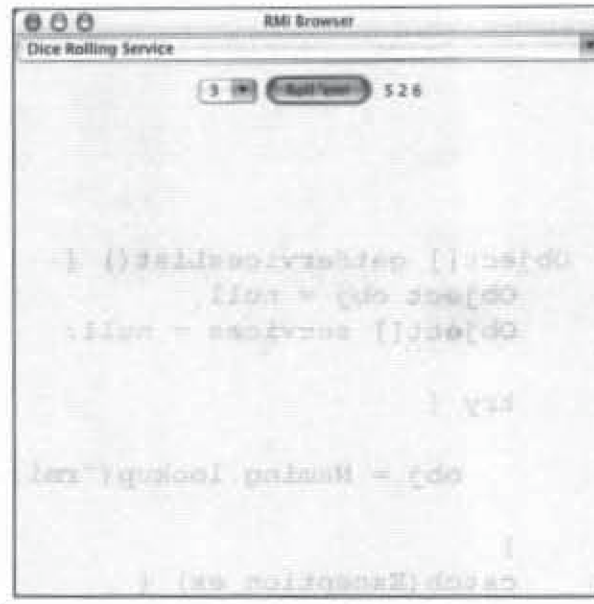
将stub转换成remote interface的类型，如此才能调用它的getServiceList()

它会返回Object的数组

用户点击JComboBox的项目后会到这里来，因此就会把相对应的服务加载

PDG

class DiceService (实现Service的通用服务)




```
import javax.swing.*;
import java.awt.event.*;
import java.io.*;

public class DiceService implements Service {

    JLabel label;
    JComboBox numOfDice;


    public JPanel getGuiPanel() {
       JPanel panel = new JPanel();
       JButton button = new JButton("Roll 'em!");
       String[] choices = {"1", "2", "3", "4", "5"};
       numOfDice = new JComboBox(choices);
       label = new JLabel("dice values here");
       button.addActionListener(new RollEmListener());
       panel.add(numOfDice);
       panel.add(button);
       panel.add(label);
       return panel;
    }

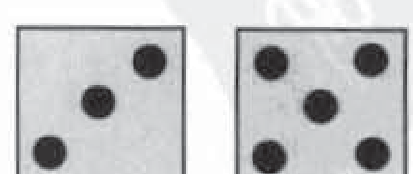
   public class RollEmListener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
         // roll the dice
         String diceOutput = "";
         String selection = (String)  numOfDice.getSelectedItem();
         int numOfDiceToRoll = Integer.parseInt(selection);
         for (int i = 0; i < numOfDiceToRoll; i++) {
            int r = (int) ((Math.random() * 6) + 1);
            diceOutput += (" " + r);
         }
        label.setText(diceOutput);

      }
    }
}
```

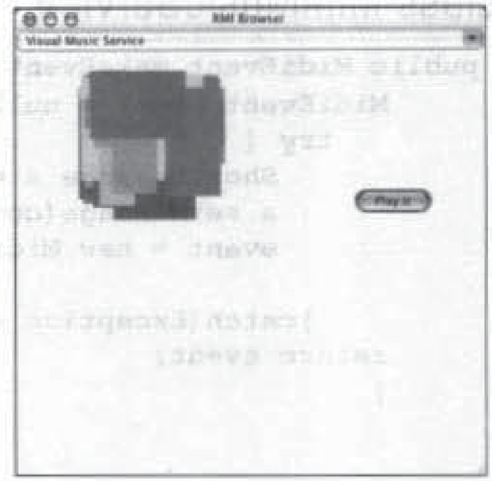
这是很重要的方法！客户端会调用这个定义在Service中的方法来创建实际的骰子

Sharpen your pencil

想想看要如何改善DiceService。一个贴心的小建议：做成图形化的骰子。



class MiniMusicService (实现Service的通用服务)



```
import javax.sound.midi.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MiniMusicService implements Service {

    MyDrawPanel myPanel;
```

就是它会显示出button并画出一堆方块

```
    public JPanel getGuiPanel() {
       JPanel mainPanel = new JPanel();
       myPanel = new MyDrawPanel();
       JButton playItButton = new JButton("Play it");
       playItButton.addActionListener(new PlayItListener());
       mainPanel.add(myPanel);
       mainPanel.add(playItButton);
       return mainPanel;
    }
```

程序代码来自第12章

```
    public class PlayItListener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {

         try {

          Sequencer sequencer = MidiSystem.getSequencer();
          sequencer.open();

          sequencer.addControllerEventListener(myPanel, new int[] {127});
          Sequence seq = new Sequence(Sequence.PPQ, 4);
          Track track = seq.createTrack();

          for (int i = 0; i < 100; i+= 4) {

              int  rNum = (int) ((Math.random() * 50) + 1);
              if (rNum < 38) {  // so now only do it if num <38 (75% of the time)
                track.add(makeEvent(144,1,rNum,100,i));
                track.add(makeEvent(176,1,127,0,i));
                track.add(makeEvent(128,1,rNum,100,i + 2));
              }
          } //结束循环

          sequencer.setSequence(seq);
          sequencer.start();
          sequencer.setTempoInBPM(220);
      } catch (Exception ex) {ex.printStackTrace();}

    } // 关闭actionperformed
   } // 关闭内部类
```

class MiniMusicService （续）

```
 public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
      MidiEvent event = null;
         try {
            ShortMessage a = new ShortMessage();
            a.setMessage(comd, chan, one, two);
            event = new MidiEvent(a, tick);

          }catch(Exception e) { }
      return event;
      }


 class MyDrawPanel extends JPanel implements ControllerEventListener {

      // only if we got an event do we want to paint
      boolean msg = false;

      public void controlChange(ShortMessage event) {
         msg = true;
         repaint();
     }

      public Dimension getPreferredSize() {
        return new Dimension(300,300);
      }

      public void paintComponent(Graphics g) {
         if (msg) {

           Graphics2D g2 = (Graphics2D) g;

           int r = (int) (Math.random() * 250);
           int gr = (int) (Math.random() * 250);
           int b = (int) (Math.random() * 250);

           g.setColor(new Color(r,gr,b));

           int ht = (int) ((Math.random() * 120) + 10);
           int width = (int) ((Math.random() * 120) + 10);

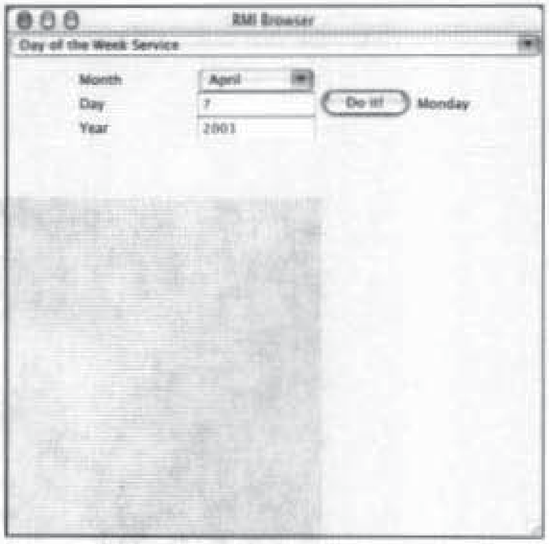
           int x = (int) ((Math.random() * 40) + 10);
           int y =  (int) ((Math.random() * 40) + 10);

           g.fillRect(x,y,ht, width);
           msg = false;

       } // close if
     } // close method
   }  // close inner class
} // close class
```

整页都跟以前是一样的

class DayOfTheWeekService (实现Service的通用服务)




```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
import java.io.*;
import java.util.*;
import java.text.*;

public class DayOfTheWeekService implements Service {

    JLabel outputLabel;
    JComboBox month;
    JTextField day;
    JTextField year;

    public JPanel getGuiPanel() {
       JPanel panel = new JPanel();
       JButton button = new JButton("Do it!");
       button.addActionListener(new DoItListener());
       outputLabel = new JLabel("date appears here");
       DateFormatSymbols dateStuff = new DateFormatSymbols();
       month = new JComboBox(dateStuff.getMonths());
       day = new JTextField(8);
       year = new JTextField(8);
       JPanel inputPanel = new JPanel(new GridLayout(3,2));
       inputPanel.add(new JLabel("Month"));
       inputPanel.add(month);
       inputPanel.add(new JLabel("Day"));
       inputPanel.add(day);
       inputPanel.add(new JLabel("Year"));
       inputPanel.add(year);
       panel.add(inputPanel);
       panel.add(button);
       panel.add(outputLabel);
       return panel;
    }

    public class DoItListener implements ActionListener {
       public void actionPerformed(ActionEvent ev) {
           int monthNum = month.getSelectedIndex();
           int dayNum = Integer.parseInt(day.getText());
           int yearNum = Integer.parseInt(year.getText());
           Calendar c = Calendar.getInstance();
           c.set(Calendar.MONTH, monthNum);
           c.set(Calendar.DAY_OF_MONTH, dayNum);
           c.set(Calendar.YEAR, yearNum);
           Date date = c.getTime();
           String dayOfWeek = (new SimpleDateFormat("EEEE")).format(date);
           outputLabel.setText(dayOfWeek);
       }
    }
}
```

创建GUI的方法 (annotation pointing to getGuiPanel())

日期格式化的工作见第10章 (annotation pointing to the SimpleDateFormat line)



恭喜你！

你办到了！

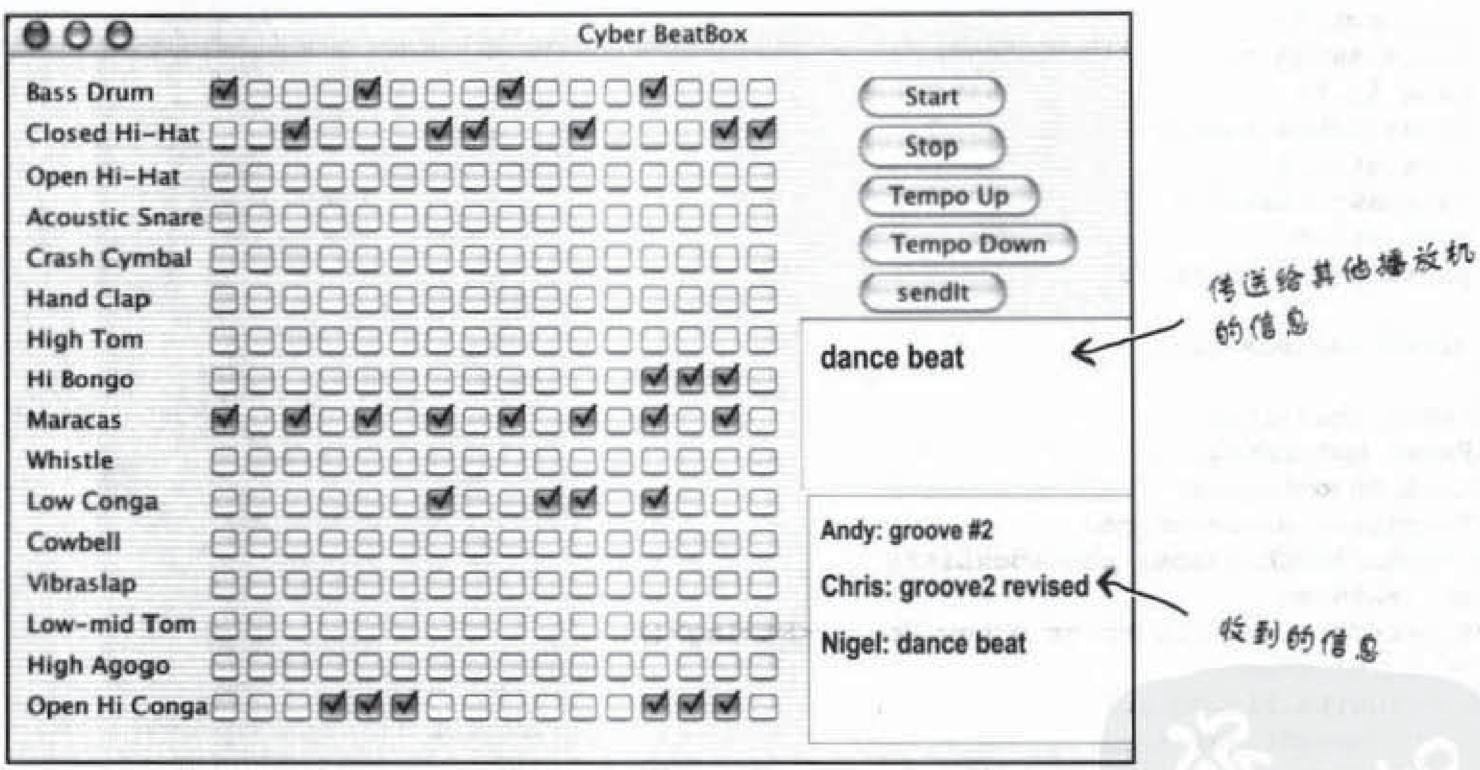
当然，后面还有两个附录。

一个索引。

一个网站……

别想偷懒。

附录A：程序料理决定版



这是 BeatBox 的完成版！

它会连接到 MusicServer 来让你传送并接收其他用户的信息以及节奏样式。

BeatBox客户端决定版

这些程序大部分都跟章节内容的程序一样，所以我们没有重复加上注释。
新增加的部分包括了：

(1) GUI：加入两个新的组件来显示收到的信息。

(2) 网络：BeatBox会连接到服务器取得输出入串流。

(3) 线程：reader这个类会持续地从服务器读取信息。

```
import java.awt.*;
import javax.swing.*;
import java.io.*;
import javax.sound.midi.*;
import java.util.*;
import java.awt.event.*;
import java.net.*;
import javax.swing.event.*;

public class BeatBoxFinal {

      JFrame theFrame;
      JPanel mainPanel;
      JList incomingList;
      JTextField userMessage;
      ArrayList<JCheckBox> checkboxList;
      int nextNum;
      Vector<String> listVector = new Vector<String>();
      String userName;
      ObjectOutputStream out;
      ObjectInputStream in;
      HashMap<String, boolean[]> otherSeqsMap = new HashMap<String, boolean[]>();

      Sequencer sequencer;
      Sequence sequence;
      Sequence mySequence = null;
      Track track;

      String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat", "Open Hi-Hat","Acoustic 
      Snare", "Crash Cymbal", "Hand Clap", "High Tom", "Hi Bongo", "Maracas", "Whistle", 
      "Low Conga", "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo" , "Open Hi Conga"};

      int[] instruments = {35,42,46,38,49,39,50,60,70,72,64,56,58,47,67,63};
```

```
public static void main (String[] args) {
   new BeatBoxFinal().startUp(args[0]);  // args[0] is your user ID/screen name
}
```

作为显示名称的命令栏参数，例如：%java BeatBoxFinal theFlash

```
public void startUp(String name) {
    userName = name;
    // open connection to the server
    try {
      Socket sock = new Socket("127.0.0.1", 4242);
      out = new ObjectOutputStream(sock.getOutputStream());
      in = new ObjectInputStream(sock.getInputStream());
      Thread remote = new Thread(new RemoteReader());
      remote.start();
    } catch(Exception ex) {
        System.out.println("couldn't connect - you'll have to play alone.");
    }
    setUpMidi();
    buildGUI();
} // close startUp
```

设置网络、输入/输出，并创建出 reader 的线程

```
 public void buildGUI() {

    theFrame = new JFrame("Cyber BeatBox");
    BorderLayout layout = new BorderLayout();
    JPanel background = new JPanel(layout);
    background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

    checkboxList = new ArrayList<JCheckBox>();

    Box buttonBox = new Box(BoxLayout.Y_AXIS);
    JButton start = new JButton("Start");
    start.addActionListener(new MyStartListener());
    buttonBox.add(start);

    JButton stop = new JButton("Stop");
    stop.addActionListener(new MyStopListener());
    buttonBox.add(stop);

    JButton upTempo = new JButton("Tempo Up");
    upTempo.addActionListener(new MyUpTempoListener());
    buttonBox.add(upTempo);

    JButton downTempo = new JButton("Tempo Down");
    downTempo.addActionListener(new MyDownTempoListener());
    buttonBox.add(downTempo);

    JButton sendIt = new JButton("sendIt");
    sendIt.addActionListener(new MySendListener());
    buttonBox.add(sendIt);

    userMessage = new JTextField();
```

GUI 程序

```
    buttonBox.add(userMessage);

    incomingList = new JList();
    incomingList.addListSelectionListener(new MyListSelectionListener());
    incomingList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    JScrollPane theList = new JScrollPane(incomingList);
    buttonBox.add(theList);
    incomingList.setListData(listVector); // no data to start with

    Box nameBox = new Box(BoxLayout.Y_AXIS);
    for (int i = 0; i < 16; i++) {
        nameBox.add(new Label(instrumentNames[i]));
    }

    background.add(BorderLayout.EAST, buttonBox);
    background.add(BorderLayout.WEST, nameBox);

    theFrame.getContentPane().add(background);
    GridLayout grid = new GridLayout(16,16);
    grid.setVgap(1);
    grid.setHgap(2);
    mainPanel = new JPanel(grid);
    background.add(BorderLayout.CENTER, mainPanel);

    for (int i = 0; i < 256; i++) {
        JCheckBox c = new JCheckBox();
        c.setSelected(false);
        checkboxList.add(c);
        mainPanel.add(c);
    } // end loop

    theFrame.setBounds(50,50,300,300);
    theFrame.pack();
    theFrame.setVisible(true);
  } // close buildGUI

  public void setUpMidi() {
     try {
     sequencer = MidiSystem.getSequencer();
     sequencer.open();
     sequence = new Sequence(Sequence.PPQ,4);
     track = sequence.createTrack();
     sequencer.setTempoInBPM(120);
   } catch(Exception e) {e.printStackTrace();}

} // close setUpMidi
```

会显示收到信息的组件

没什么不一样的程序

创建Track

```
public void buildTrackAndStart() {
   ArrayList<Integer> trackList = null; // this will hold the instruments for each
   sequence.deleteTrack(track);
   track = sequence.createTrack();

   for (int i = 0; i < 16; i++) {

     trackList = new ArrayList<Integer>();

     for (int j = 0; j < 16; j++) {
        JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
        if (jc.isSelected()) {
           int key = instruments[i];
           trackList.add(new Integer(key));
        } else {
          trackList.add(null);  // because this slot should be empty in the track
        }
     } // close inner loop
     makeTracks(trackList);
   } // close outer loop
   track.add(makeEvent(192,9,1,0,15)); // - so we always go to full 16 beats
   try {
      sequencer.setSequence(sequence);
      sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
      sequencer.start();
      sequencer.setTempoInBPM(120);
  } catch(Exception e) {e.printStackTrace();}
 } // close method

public class MyStartListener implements ActionListener {
   public void actionPerformed(ActionEvent a) {
      buildTrackAndStart();
   } // close actionPerformed
} // close inner class

public class MyStopListener implements ActionListener {
   public void actionPerformed(ActionEvent a) {
      sequencer.stop();
  } // close actionPerformed
} // close inner class

public class MyUpTempoListener implements ActionListener {
   public void actionPerformed(ActionEvent a) {
      float tempoFactor = sequencer.getTempoFactor();
      sequencer.setTempoFactor((float)(tempoFactor * 1.03));
  } // close actionPerformed
} // close inner class
```

跟以前的章节内容一样

GUI的监听者

```
    public class MyDownTempoListener implements ActionListener {
        public void actionPerformed(ActionEvent a) {
          float tempoFactor = sequencer.getTempoFactor();
          sequencer.setTempoFactor((float)(tempoFactor * .97));
        }
    }

    public class MySendListener implements ActionListener {
       public void actionPerformed(ActionEvent a) {
          // make an arraylist of just the STATE of the checkboxes
           boolean[] checkboxState = new boolean[256];
          for (int i = 0; i < 256; i++) {
             JCheckBox check = (JCheckBox) checkboxList.get(i);
             if (check.isSelected()) {
               checkboxState[i] = true;
             }
          } // close loop
        String messageToSend = null;
        try {
           out.writeObject(userName + nextNum++ + ": " + userMessage.getText());
           out.writeObject(checkboxState);
        } catch(Exception ex) {
            System.out.println("Sorry dude. Could not send it to the server.");
        }
        userMessage.setText("");
      } // close actionPerformed
    } // close inner class

    public class MyListSelectionListener implements ListSelectionListener {
        public void valueChanged(ListSelectionEvent le) {
          if (!le.getValueIsAdjusting()) {
             String selected = (String) incomingList.getSelectedValue();
             if (selected != null) {
                // now go to the map, and change the sequence
                 boolean[] selectedState = (boolean[]) otherSeqsMap.get(selected);
                 changeSequence(selectedState);
                 sequencer.stop();
                 buildTrackAndStart();
             }
          }
        } // close valueChanged
  } // close inner class
```

文本信息和节拍样式会被序列化送到输出串流

用户点选信息时会马上加载节拍样式并开始播放

```
public class RemoteReader implements Runnable {
    boolean[] checkboxState = null;
    String nameToShow = null;
    Object obj = null;
    public void run() {
      try {
        while((obj=in.readObject()) != null) {
          System.out.println("got an object from server");
          System.out.println(obj.getClass());
          String nameToShow = (String) obj;
          checkboxState = (boolean[]) in.readObject();
          otherSeqsMap.put(nameToShow, checkboxState);
          listVector.add(nameToShow);
          incomingList.setListData(listVector);
        } // close while
      } catch(Exception ex) {ex.printStackTrace();}
    } // close run
} // close inner class

public class MyPlayMineListener implements ActionListener {
   public void actionPerformed(ActionEvent a) {
      if (mySequence != null) {
         sequence = mySequence;  // restore to my original
      }
   } // close actionPerformed
} // close inner class

public void changeSequence(boolean[] checkboxState) {
   for (int i = 0; i < 256; i++) {
       JCheckBox check = (JCheckBox) checkboxList.get(i);
        if (checkboxState[i]) {
             check.setSelected(true);
        } else {
              check.setSelected(false);
        }
    } // close loop
} // close changeSequence

public void makeTracks(ArrayList list) {
   Iterator it = list.iterator();
   for (int i = 0; i < 16; i++) {
      Integer num = (Integer) it.next();
      if (num != null) {
         int numKey = num.intValue();
         track.add(makeEvent(144,9,numKey, 100, i));
         track.add(makeEvent(128,9,numKey,100, i + 1));
      }
    } // close loop
 } // close makeTracks()
```

线程执行的任务

收到的信息会加入JList组件

MIDI部分与以前的内容完全一样

```
    public  MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
        MidiEvent event = null;
        try {
          ShortMessage a = new ShortMessage();
          a.setMessage(comd, chan, one, two);
          event = new MidiEvent(a, tick);
        }catch(Exception e) { }
        return event;
    } // close makeEvent

} // close class
```

也跟以前一样

Sharpen your pencil

有哪些方法可以改善这个程序？

你可以从以下的方向考虑：

(1) 加载新的样式时，目前的样式就会消失。如果正在设计新样式，则辛苦都白费了。你也许会希望有个提示对话框确认是否要先存盘。

(2) 输入不正确的命令栏参数会导致异常！把程序改成在这种情况下能够使用默认值或给使用者一个提示，尽可能地让程序优雅的运行而不会粗鲁得直接挂掉。

(3) 随机产生节奏样式（译注：这是个烂主意，相信我，计算机只能随机产生噪音）。

BeatBox服务器端最终版

大部分的程序与讨论网络以及线程的章节内容一样。唯一的差别是服务器接收并传送两个对象而不是一个对象。

```
import java.io.*;
import java.net.*;
import java.util.*;

public class MusicServer {

    ArrayList<ObjectOutputStream> clientOutputStreams;

    public static void main (String[] args) {
         new MusicServer().go();
    }

    public class ClientHandler implements Runnable {

         ObjectInputStream in;
         Socket clientSocket;

         public ClientHandler(Socket socket) {
           try {
             clientSocket = socket;
              in = new ObjectInputStream(clientSocket.getInputStream());

           } catch(Exception ex) {ex.printStackTrace();}
          } // close constructor


        public void run() {
             Object o2 = null;
             Object o1 = null;
           try {

             while ((o1 = in.readObject()) != null) {

                o2 = in.readObject();

                System.out.println("read two objects");
                tellEveryone(o1, o2);
              } // close while

           } catch(Exception ex) {ex.printStackTrace();}
        } // close run
    } // close inner class
```

```
 public void go() {
   clientOutputStreams = new ArrayList<ObjectOutputStream>();

      try {
        ServerSocket serverSock = new ServerSocket(4242);

        while(true) {
           Socket clientSocket = serverSock.accept();
           ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream());
           clientOutputStreams.add(out);

           Thread t = new Thread(new ClientHandler(clientSocket));
           t.start();

           System.out.println("got a connection");
        }
     }catch(Exception ex) {
       ex.printStackTrace();
     }
 } // close go

 public void tellEveryone(Object one, Object two) {
    Iterator it = clientOutputStreams.iterator();
    while(it.hasNext()) {
      try {
         ObjectOutputStream out = (ObjectOutputStream) it.next();
         out.writeObject(one);
         out.writeObject(two);
       }catch(Exception ex) {ex.printStackTrace();}
     }
 } // close tellEveryone

} // close class
```

附录B

十大遗珠之憾



我们已经说了很多，你也差不多快看完了。我们会很想念你的，但在让你上场前如果漏掉一些东西没有告诉你，我们会很不放心。可是附录里面放不了这么多东西，实际上我们曾经尝试要把所有东西挤到这本书里面。最后结果是字体要缩小到0.00003英寸才办得到，这根本不能读，所以我们决定放弃这个念头。最后我们留下了这10项最重要的主题。

#10 位操作

你在乎吗？

我们之前讨论过byte有8位，short有16位。也许你会需要个别的操作某个位的值，也许有一天你会遇到Java烤面包机并发现它只有1KB的内存，为了节省空间只好用个别字节来存储设定。

按位非运算符：~

这个运算符会将primitive主数据类型的字节组合值取反。

```
int x = 10;          // 00001010
x = ~x;              // 变成 11110101
```

接下来的3个运算符会逐位地比较primitive主数据类型的值，然后返回比较结果，下面是要比较的范例。

```
int x = 10;     // 位组合是00001010
int y =  6;     // 位组合是00000110
```

按位与运算符：&

如果两个位都是1才会返回1，否则返回0。

```
int a = x & y;  // 位组合是00000010
```

按位或运算符：|

只要其中有一个是1就会返回1。

```
int a = x | y;  // 位组合是00001110
```

按位异或运算符：^

位相同时返回0，否则返回1。

```
int a = x ^ y;  // 位组合是00001100
```

移位运算符

此运算符取用单一primitive主数据类型并向某个方向执行移位，如果你的二进制运算技巧够好的话，你就会发现左移的效果与乘以 2 是一样的，而右移跟除以 2 一样。

接下来的3个运算符使用下面这个值。

```
int x = -11;    //      位组合是 11110101
```

以下将全世界最短的负数表示成补码。整数最左方的字节称为符号位。Java中的负整数此位永远是1。正数此位永远是0。Java使用补码来存储负值，改变正负号时要把位取反然后加1。

右移运算符：>>

以指定量右移字节合，左方补0，正负号不变。

```
int y = x >> 2;   // 位组合是11111101
```

无符号右移运算符：>>>

与>>一样，但第一个位也会补0，正负号可能会改变。

```
int y = x >>> 2;   // 位组合是00111101
```

左移运算符：<<

与>>>一样，但方向相反，右方补0，正负号可能改变。

```
int y = x << 2;   // 位组合是11010100
```

#9 不变性

为什么要在乎String的不变性？

当程序越来越大时，不可避免地会有很多 String 对象。为了安全性和节省空间（例如在手机上执行）的原因，String 是不变的。意思是下面的程序：

```
String s = "0";
for (int x = 1; x < 10; x++) {
  s = s + x;
}
```

实际上会创建出10个String对象（“0”，“01”，“012”……“0123456789”）。最后会引用到“0123456789”这个值，但此时会存在10个 String。

创建新的String 时，Java虚拟机会把它放到称为“String Pool”的特殊存储区中。如果已经出现同值的String，Java虚拟机不会重复建立String，只会引用已存在者。这是因为String是不变的，引用变量无法改变其它参考变量引用到的同一个String值。

String pool不受Garbage Collector管理，因此我们在for循环中建立的10个String有9个是在浪费空间。

这要如何节省内存？

如果你不注意的话就不会！但如果你知道String的不变性，就可以利用它来节省空间。如果要执行一堆String操作，则StringBuilder这个稍后会介绍的class更为合适。

为什么要在乎包装类的不变性？

之前我们有讨论到包装类的两个主要用途：

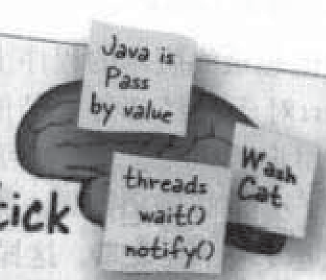
（1）将 primitive主数据类型包装成对象。

（2）使用静态的工具方法（例如说Integer.parseInt()）。

当你创建像下面这个包装对象时：

```
Integer iWrap = new Integer(42);
```

它的值永远会是 42。包装对象没有 setter。你当然还是能够让 iWrap 引用别的包装对象，但是会产生两个对象。包装对象创建后就无法改变该对象的值！



Make it Stick

玫瑰是红的，Keroro是绿的。
String不可变，包装类也是。

这段提示是免费送给你的贴心小礼物。

#8 断言

我们并没有对开发过程的除错进行很多的讨论。我们认为你应该从命令行来学习Java，就像这本书的设计方式一样。一旦你成为Java大师之后，或许你会使用IDE*的debug工具。以前的Java程序设计师得在程序中加入一大堆System.out.println()命令来显示除错的信息，列出变量值，还有执行到那里的信息以观察流程控制的走向。等到程序可以正确执行的时候，还要看过所有的程序把println()拿掉。这很麻烦且容易出错。从Java 1.4之后除错就变得容易多了。为什么？

断言

它就像是喝了3瓶蛮牛加上两粒克补的println()一样。使用的方式跟加入println()差不多，Java 5.0的编译器会假设所编译的源文件与5.0兼容，此时预设断言是打开的。

执行时，如果没有特别设定的话，被加入到程序中的assert命令会被Java虚拟机忽略。但若你指定Java虚拟机要打开断言的话，它就能够在不变动任何一行程序的情况下帮助你对程序除错。

有些人会抱怨最终版本的程序上还有assert命令，然而留着assert对于已经部署安装的程序代码是很有价值的。如果用户遇到问题，你就可以指示客户打开断言来执行程序，并取得输出结果。如果没有留下assert，你就很难知道发生了什么事。这样做没什么坏处；未打开时，Java虚拟机会忽略掉它们，所以不会影响性能。

如何使用断言

在你认为一定是true的地方加上assert命令，例如：

```
assert (height > 0);
```

```
// 若true，则继续执行
// 若false，抛出AssertionError
```

你也可以加上一点信息：

```
assert (height > 0) : "height = " + height +
" weight = " + weight;
```

在冒号后面的指令可以是任何解出非null值的合法Java语句。无论如何千万别在assert中改变对象的状态！不然的话，打开assertion执行时可能会改变程序的行为。

带有断言的编译和执行

编译：

```
javac TestDriveGame.java
```

（注意并不需要特殊选项）

执行：

```
java -ea TestDriveGame
```

*IDE代表Integerated Development Environment（集成开发环境），例如Eclipse、Borland的JBuilder或开放源码的NetBeans。

#7 块区域

我们在第9章讨论局部变量的生命周期范围只有在声明它的方法还待在栈上的期间内。但某些变量的生命周期会更短。我们经常会在方法中建立程序区段，但是还没有讨论过块的议题。通常区段程序代码会在方法中，并以{ }字符符号来区分。常见的例子有循环（for与 while）以及条件测试运算（如if）。

以下面的程序为例：

```
void doStuff() {
    int x = 0;
    for(int y = 0; y < 5; y++) {

        x = x + y;
    }

    x = x * y;
}
```

方法区段的起点

生命周期跟整个方法一样的局部变量

for循环的block，y的范围只限于此区段

使用x和y不是问题

区段终点

啊！不能编译，y已经超出范围了（注意：某些其他语言容许此状况）

method区段的终点，x也不能超出此范围

在上面的范例中，y是个块方法，被宣告在块中，一离开循环就超出范围。使用局部变量比实体变量更方便除错且扩充性更好，而能用块变量时就把局部变量换掉。编译器会确保你不会用到超出范围的变量，因此不用担心执行时会出错。

#6 链接的调用

本书中很少这么使用，因为我们尝试让语法尽可能的干净和方便阅读。然而在Java中有很多合法的快捷方式，毫无疑问，你会看到很多如此编写的程序代码。你最有可能遇到的结构之一就是链接调用。像下面这样：

```
StringBuffer sb = new StringBuffer("spring");
sb = sb.delete(3,6).insert(2,"umme").deleteCharAt(1);
System.out.println("sb = " + sb);
// 结果是 sb = summer
```

第二行程序是怎么回事？好吧，这是很刻意弄出来的例子，但你需要知道如何解释它们。

(1) 从左开始看到右边。

(2) 找出最左方的方法，此例为sb.delete(3,6)。如果查询StringBuffer的文件，你会看到delete()返回StringBuffer对象。结果就是delete()返回一个值为“spr”的StringBuffer对象。

(3) 接着是对新创建出来的StringBuffer对象调用insert()，此调用也会返回StringBuffer对象，然后如此这般的下去直到最右边。理论上，你在同一行程序可以无限地链接下去（实际上很少会超过3个）。如果不这么做，上面的第二行程序可以改写成下面这种更易于阅读的形式：

```
sb = sb.delete(3,6);
sb = sb.insert(2,"umme");
sb = sb.deleteCharAt(1);
```

下面有个更常见、更有用的例子，你之前就已经看到我们用过，现在又被带出来。这是个调用方法又不需维持一个引用的方法。

```
class Foo {
   public static void main(String [] args) {
     new Foo().go();
   }
   void go() {
    // 真正要执行的程序
   }
}
```

← 我们需要调用go()，但又不需要维持一个对Foo的引用

#5 Anonymous和Static Nested Classes

嵌套的类有很多种

以前提到GUI事件处理时，我们有使用内部（套迭）的类来解决接听的实现。这是内部类很常见、实用，且易于阅读的形式，类是包在另一个类的括号中。要记住这代表你需要外部类的实例才能取得内部类的实例，因为内部是外部的一个成员。

但还有包括了static与anonymous等类型的内部类。我们不会详谈细节，但还是要能让你看别人的程序时懂这种语法。因为Java几乎能做任何事情，所以也许没有其它东西能够产生出更可怕的不具名内部类。我们先从比较简单的静态嵌套类开始。

静态嵌套类

你已经知道static是什么意思——跟在类而不是特定实例的东西。静态嵌套的类看起来跟我们用在事件监听者的非静态类很像，但被标示为static的。

静态嵌套类就是被包在另外一个类里面标记为static的类

```
public class FooOuter {
  static class BarInner {
     void sayIt() {
        System.out.println("method of a static inner class");
     }
  }
}

class Test {
  public static void main (String[] args) {
     FooOuter.BarInner foo = new FooOuter.BarInner();
     foo.sayIt();
  }
}
```

因为是静态的，所以不需外层的实例

静态嵌套很像一般非嵌套的，他们并未与外层对象产生特殊关联。但因为还被认为是外层的一个成员，所以能够存取任何外层的私用成员。然而只限于也是静态的，这是因为静态并没有实例。

nested与inner的差别

任何被定义在另一个类范围内的类被称为嵌套的类，不管是否为匿名、静态、正常或其他类型。只要是在另一个类中，技术上它就被认定是嵌套的类。但非静态的嵌套的类通常被称做内部的类，我们之前也是这么写的。基本上所有内部类都是嵌套的，但不是所有嵌套类都是内部的。

匿名的内部类

假设你在编写GUI程序，突然发现你需要一个有实现ActionListner的类的实例，你也发现你没有任何该实例，其实你根本没有写过这样的类（译注：听起来很白痴，实际上很常见），此时你有两种选择：

（1）在程序中编写内部类，就像我们之前的GUI程序一样。

或者

（2）当场创建出匿名的内部类。就在现场开始写一个类出来。没错，就在需要传入一个实例的地方创建出这个类。这代表你把整个类当作参数传进去。

```
import java.awt.event.*;
import javax.swing.*;
public class TestAnon {
   public static void main (String[] args) {

      JFrame frame = new JFrame();
      JButton button = new JButton("click");
      frame.getContentPane().add(button);
      // button.addActionListener(quitListener);

      button.addActionListener (new ActionListener() {
        public void actionPerformed(ActionEvent ev) {
          System.exit(0);
        }
      });

   }
}
```

创建框架并加入按钮，此时需要注册按钮的监听者

通常是传入一个内部类的实例

但是我们也可以把整个类的定义传进去，此语法也会自动地创建出该类的实例

这个语句

注意到此处的ActionListner其实是个接口，而我们不能创建接口的实例！但这种情况的语法就是要这样

一直到这里才算完整

#4 存取权限和存取修饰符(谁可以看到什么)

Java有4种存取权限等级与3种存取修饰符。只有3种修饰符的原因是因为还有一个缺省的（不加任何的修饰字）的权限等级。

存取权限（从限制最少的开始列出）

public ← 代表任何程序代码都可以存取的公开事物（类、变量、方法、构造函数等）

protected ← 受保护的部分运行起来像是default，但也能允许不在相同包的子类继承受保护的部分

default ← 只有在同一包中的默认事物能够存取

private ← 只有同一类中的程序代码才能存取，它是对类而不是对象设限，所以Dog可以看到别的Dog的私用部分，但Cat就不能看到Dog的私用部分

存取修饰符

```
public
protected
private
```

通常你只会用到 public和private 两种等级。

public

用public来设定打算要开放给其他程序代码的类、常数（static final变量）、方法，以及大部分的构造函数。

private

设定给大部分的实例变量，以及不想开放给外部程序调用的方法。

虽然你可能不会用到其余两个，你还是必须知道它们的作用。

#4 存取权限和存取修饰符（续）

default与protected

default

protected与default权限等级都是作用在包上。默认的等级是最简单的，它代表只有同一包也是默认等级的程序可以存取。例如没有被明确声明为public的类只能被同一包中的类存取。

存取又是什么意思呢？不能存取某类的程序代码是不能想到该类的。这个想字代表使用。例如说你不能初始化或声明此受限制类型的变量、参数、返回值。反正就是不能在程序中提到这个类！不然编译器会跟你解释概念。

想象一下这样隐含的意义——在默认的类中的public方法意味着此方法并不全然是public的。如果你无法存取该类就无法取用此方法。

为什么要限制同一包的存取？通常包是一群在运作上有关联的类。所以互相存取程序代码是很合理的，而只有少部分的类与方法会开放给外部。

这就是default。它是如此的单纯——只有同一包可以存取。

protected

它与默认的等级几乎一样，只有一处不同：允许不同包中的子类继承它的成员。这就是protected 所带来的功能——不在同一个包的子类。

大部分的开发者很少有机会动用到protected，也许有一天你会遇到适用的设计和时机。protected 只能应用在继承上。如果不同包的子类有对父类实例的引用，子类无法透过此引用存取父类的 protected方法！唯一的办法只能通过继承取得此方法。

#3 String和StringBuffer/StringBuilder的方法

Java API中最常用到的类包括了String和StringBuffer（因为String是不变的，使用StringBuffer/StringBuilder来操作String比较有效率）。从Java 5.0起，你应该以StringBuilder取代StringBuffer，除非这些操作必须在不常见的thread安全环境中进行。下面列出一些比较重要的方法：

```
char charAt(int index);                 // 字符出现的位置
int length();                           // 字符串有多长
String substring(int start, int end);   // 抽出一部分
String toString();                      // 此物的String表示值
```

连接字符串：

```
String concat(string);                  // String使用
String append(String);                  // StringBuffer和StringBuilder使用
```

String的方法：

```
String replace(char old, char new);     // 替换
String substring(int begin, int end);   // 抽出一部分
char [] toCharArray();                  // 转换成char数组
String toLowerCase();                   // 转成小写
String toUpperCase();                   // 转成大写
String trim();                          // 删除后端空格符
String valueOf(char [])                 // 转换成String
String valueOf(int i)                   // 转换成String
                                        // 也有其他primitive主数据类型版本
```

StringBuffer和StringBuilder的方法：

```
StringBxxxx delete(int start, int end);                           // 删除部分
StringBxxxx insert(int offset, any primitive or a char []);       // 插入
StringBxxxx replace(int start, int end, String s);                // 取代
StringBxxxx reverse();                                            // 翻转
void setCharAt(int index, char ch);                               // 替换字符
```

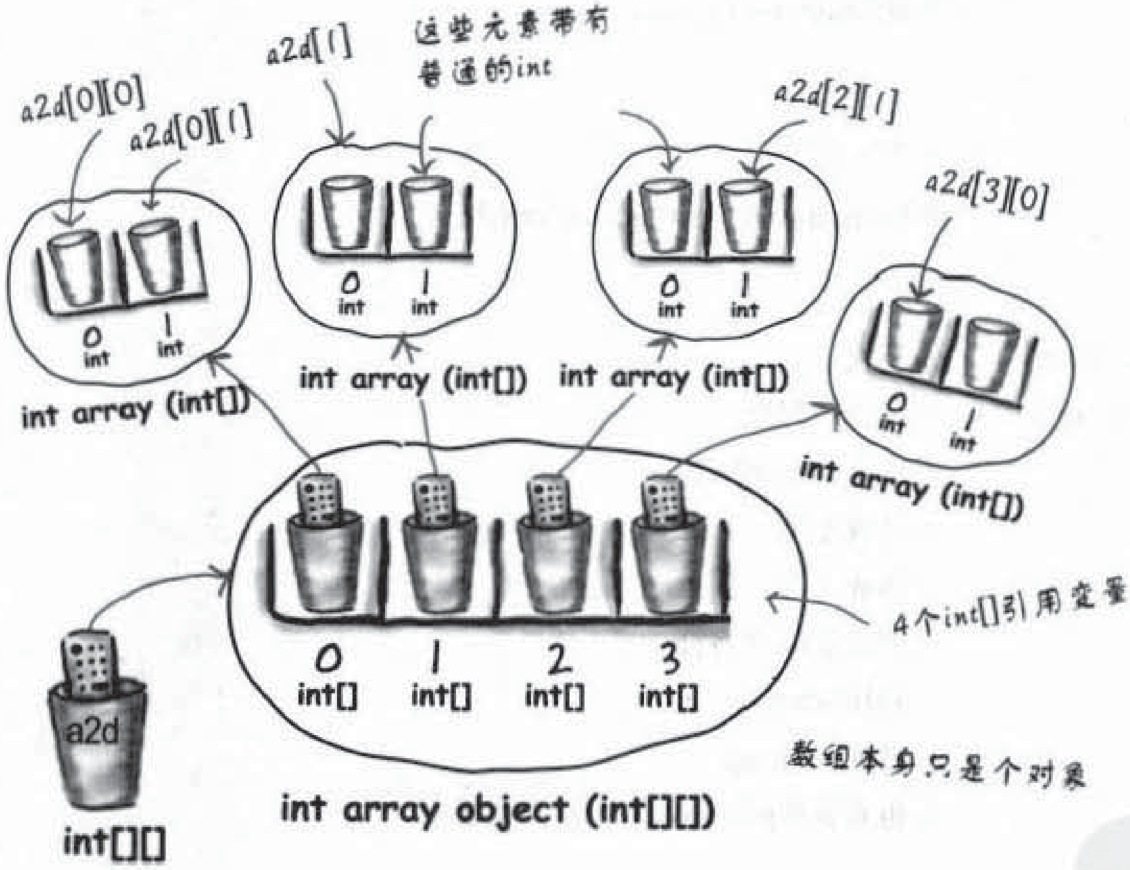
注意：Bxxxx 不是在骂人

#2 多维数组

在大部分的语言中，如果你创建了4×2的二维数组，看起来会像是个有8个元素的4×2方阵。但在Java中，此数组实际上是由5个数组连接而成！在Java中，二维数组只是个数组的数组（三维数组是数组的数组的数组）：

```
int[][] a2d  = new int [4][2];
```

上面程序会让Java虚拟机创建出有4个元素的数组，这些元素实际上是对2个元素数组的引用变量。



操作多维数组：

（1）存取第三个数组的第二个元素： `int x = a2d[2][1];`

（2）对某个子数组创建引用： `int[] copy = a2d[1];`

（3）初始化2×3数组： `int[][] x = {{2,3,4}, {7,8,9}};`

（4）创建非常规二维数组：

```
int[][] y = new int [2][];      //创建长度为2的第一层
y[0] = new int [3];      // 创建三元素的子数组
y[1] = new int [5];      // 创建五元素的子数组
```

我们还没有讨论到的第一大主题是……

#1 枚举（又称为枚举类型或Enum）

我们以前讨论过定义在API中的常量，例如JFrame.Exit_ON_CLOSE。你也可以用标记为static final的变量来设定常量。有时你会需要一组常量来代表可用值的集合。这样的一组可用值被称为枚举。在Java 5.0中你可以创建让旧版程序员羡慕的要死的全功能货真价实枚举。

团员有谁？

假设你在帮你最喜欢的乐团架构网站，且想要确保所有的讨论都会指向特定的团员。

古老的方法：

```
public static final int JERRY = 1;
public static final int BOBBY = 2;
public static final int PHIL = 3;

// 稍后

if (selectedBandMember == JERRY) {
  // 执行与JERRY有关的工作
}
```

我们祈祷到时候这个值是合理有效的！

好消息是这样的，程序有很高的可读性。另外一项好消息是你无法变更已经创建出来的常数值，JERRY永远会是1。坏消息是你没有办法能够简单地确保selectedBandMember的值一定是1，2，3。如果有个程序不小心把selectedBandMember设成9527，你的程序很可能就因此无法运行了……

#1 枚举还没完……

同样的情况套在Java 5.0的enum上面就很简单了，因为这是很基本的枚举情况，所以会很简单。

全新正式的enum是这样的：

这很像类的定义不是吗？enum就是一种特殊的类

```
public enum Members { JERRY, BOBBY, PHIL };
public Members selectedBandMember;

// 稍后

if (selectedBandMember == Members.JERRY) {
  // 执行与JERRY有关的工作
}
```

Members类型的selectedBandMember只能有JERRY，BOBBY，PHIL这3种值

无需担心此变量的值

引用到enum的实例

enum会继承java.lang.Enum

在创建enum时，其实是隐含地继承java.lang.Enum来创建新的类。你可以在独立的源文件中把enum宣告成独立的类，或是作为其他类的一员。

使用if或switch

我们可以在if或switch中使用enum。注意到我们能够以==或.equals()来比较enum的实例。一般认为==是比较好的风格。

赋enum值

```
Members n = Members.BOBBY;
if (n.equals(Members.JERRY)) System.out.println("Jerrrry!");
if (n == Members.BOBBY) System.out.println("Rat Dog");

Members ifName = Members.PHIL;
switch (ifName) {
  case JERRY: System.out.print("make it sing ");
  case PHIL: System.out.print("go deep ");
  case BOBBY: System.out.println("Cassidy! ");
}
```

这两种方式都行

猜猜看会有什么输出

答案：go deep Cassidy!

#1 枚举，说完了

相同enum的特殊版本

你可以在enum中加入构造函数、方法、变量和特定常量的内容（class body）。这不是很常见，但有时你还是会遇到：

```
public class HfjEnum {

  enum Names {
    JERRY("lead guitar") { public String sings() {
                     return "plaintively"; }
              },
    BOBBY("rhythm guitar") { public String sings() {
                     return "hoarsely"; }
              },
    PHIL("bass");

    private String instrument;

    Names(String instrument) {
      this.instrument = instrument;
    }
    public String getInstrument() {
      return this.instrument;
    }
    public String sings() {
      return "occasionally";
    }
  }

  public static void main(String [] args) {
    for (Names n : Names.values()) {
      System.out.print(n);
      System.out.print(", instrument: "+ n.getInstrument());
      System.out.println(", sings: " + n.sings());
    }
  }
}
```

传给下方定义构造函数的参数

特定常量的内容。把它们当作是基本enum方法的覆盖（此例中是sings()）

这是enum的构造函数，会对每个被声明的enum值执行一次（此例中运行3次）

会被main()调用

每个enum都有内置的values()，通常会用在for循环

```
File Edit Window Help Bootleg
%java HfjEnum
JERRY, instrument: lead guitar, sings: plaintively
BOBBY, instrument: rhythm guitar, sings: hoarsely
PHIL, instrument: bass, sings: occasionally
%
```

基本的sings()只会在enum值没有特定的内容时才会被调用



迷你推理短篇

归乡路

星际战舰“Traverser”号的魏元祖船长收到一份总部传来的紧急最高机密信息。信息带有30个高度加密编码过的导航码，只有这些导航码指出的路径能带他们通过危机四伏的敌方控制区。来自邻近星系的骇克人有着高度进化的科技，能够发送捣蛋射线潜入“Traverser”号上面唯一的导航计算机，在计算机的堆空间埋进伪造的对象。此外，他们还有种技术能够偷偷地将引用变量改指向伪造对象。“Traverser”号的船员手上只有一种工具能够抵抗骇克人对导航系统的Java 5.0程序代码做邪恶的攻击，就是程序代码内嵌的病毒检测器。



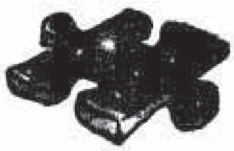
魏船长以下面列出的命令要求史正浩处理关键程序代码：

“将前五个导航码放进ParsecKey类型的数组中。其余25个放进5 × 5的QuadrantKey类型二维数组中。将这两个数组传入public final的ShipNavigation这个class的plotCourse()中。一旦取回路线对象就对所有的引用变量执行病毒检查以及NavSim仿真程序，然后向我报告结果”。

几分钟后史正浩拿到了模拟输出。“模拟结果出来了”。“很好”，船长回应，“请说”。“是，长官”，史正浩继续说明，“我先以ParsecKey[] p = new ParsecKey[5]; 声明并创建ParsecKey类型的数组，接着是QuadrantKey[][] q = new QuadrantKey[5][5]; 再来使用for循环加载5个导航码给ParsecKey数组，同样使用for循环加载25个导航码给QuadrantKey数组。此后我用病毒检测器对32个引用变量、ParsecKey数组和它的5个元素、QuadrantKey数组和它的25个元素做检查。等待检测器报告没有发现病毒时，我就运行了仿真程序并重新执行病毒检测”。

魏元祖船长瞪着史正浩看着，冷冷地说：“小史，因为你危害这艘船，我要关你禁闭，我不想再看到你进到指挥舱，除非你把Java学好！布尔中尉，你接手这项任务”。

为什么船长要关他禁闭呢？



迷你推理短篇解答



归乡路

魏船长知道在Java中，多维数组实际上是数组的数组。5 × 5的QuadrantKey数组实际上总共有31个引用变量能够存取所有的元素：

1个引用变量： q

5个引用变量：q[0] 到 q[4]

25个引用变量：q[0][0] 到 q[4][4]

共计31个。

史正浩忘记算到埋在q数组的5个一维数组。如果骇客人通过这5个引用变量来潜入，则病毒检测不会捕获到这个危机。

索引

符号和标识

&, &&, |, ||（布尔运算符）151, 660

&, <<, >>, >>>, ^, |, ~（位运算符）660

++ --（递增/递减）105, 115

+（String连接运算符）17

.（圆点运算符）36
  引用 54

<, <=, ==,!=, >, >=（比较运算符）86, 114, 151

<, <=, ==, >, >=（比较运算符）11

A

abandoned objects（见垃圾收集器）

abstract（抽象）
  class（类）200–210
  class modifier（类修饰符）200

abstract methods（抽象方法）
  declaring（声明）203

access（存取）
  and inheritance（继承）180
  class modifiers（类修饰符）667
  method modifiers（方法修饰符）81, 667
  variable modifiers（变量修饰符）81, 667

accessors and mutators（见getters和setters）

ActionListener interface（ActionListener接口）358, 358–361

addActionListener() 359–361

advice guy（叶教授）480, 484

Aeron™ 28

animation（动画效果）382–385

API（应用编程接口）154–155, 158–160
  ArrayList 532
  collections（集合）558

appendix A（附录A）649–658
  beat box final client（beat box客户端最终版）650
  beat box final server（beat box服务器端最终版）657

appendix B（附录B）
  access levels and modifiers（存取权限和存取修饰符）667
  assertions（断言）662
  bit manipulation（位操作）660
  block scope（区块域）663
  immutability（不变性）661
  linked invocations（链接的调用）664
  multidimensional arrays（多维数组）670
  String and StringBuffer methods（String和StringBuffer方法）669

apples and oranges（苹果和桔子）137

arguments（参数）
  method（方法）74, 76, 78
  polymorphic（多态）187

ArrayList 132, 133–138, 156, 208, 558
  API（应用编程接口）532
  ArrayList<Object> 211–213
  autoboxing（autoboxing）288–289
  casting（转换）229

arrays（数组）
  about（介绍）17, 59, 135
  assigning（赋值）59
  compared to ArrayList（比较ArrayList和一般数组）134–137
  creation（创建）60
  declaring（声明）59
  length attribute（长度）17
  multidimensional（多维）670
  objects, of（对象）60, 83
  primitives, of（primitives主数据类型）59

assertions（断言）
  assertions（断言）662

assignments, primitive（赋值，primitive主数据类型）52

assignments, reference variables（赋值，引用变量） 55, 57, 83

atomic code blocks（原子块）510–512（见线程）

audio.（见midi）

autoboxing（autoboxing）288–291
  and operators（运算符）291
  assignments（赋值）291

B

bark different（不同的吠声）73

bathtub（澡盆）177

beat box（beat box）316, 347, 472（见附录A）

beer（啤酒）14

behavior（行为）73

Bela Fleck（Bela Fleck）30

bitwise operators（位运算符）660

bit shifting（移位）660

block scope（块区域）663

boolean（布尔）51

boolean expressions（布尔表达式）11, 114
  logical（逻辑）151

BorderLayout manager（BorderLayout管理器） 370–371, 401, 407

BoxLayout manager（BoxLayout管理器）411

brain barbell（brain barbell）33, 167, 188

break statement（break语句）105

BufferedReader 454, 478

BufferedWriter 453

buffers（缓冲区）453, 454

byte（字节）51

bytecode（字节码）2

C

Calendar 303–305
  methods（方法）305

casting（转换）
  explicit primitive（显primitive主数据类型）117
  explicit reference（显引用）216
  implicit primitive（隐primitive主数据类型）117

catching exceptions（捕获异常）326
  catch（catch）338
  catching multiple exceptions（捕获多重异常）329, 330, 332
  try（try）321

catch blocks（catch块）326, 338
  catching multiple exceptions（捕获多重异常）329, 330, 332

chair wars（椅子大战）28, 166

char（char）51

chat client（聊天室客户端程序）486
  with threads（线程）518

chat server (simple)（聊天服务器程序）520

checked exceptions（检查异常）
  runtime vs.（运行期间）324

checking account（见Ryan和Monica）

check box (JCheckBox)（复选框）416

class（类）
  abstract（抽象）200–210
  concrete（创建）200–210
  designing（设计）34, 41, 79
  final（final）283
  fully qualified names（完整名称）154–155, 157

client/server（客户端/服务器端）473

code kitchen（程序料理）
  beat box save and restore（beat box保存和恢复） 462
  final beat box.（beat box最终版，见附录A）
  making the GUI（创建GUI）418
  music with graphics（带有图形的音乐）386
  playing sound（播放声音）339

coffee cups（咖啡杯）51

collections（集合）137, 533
  API（应用程序接口）558
  ArrayList（ArrayList）137
  ArrayList<Object>（ArrayList<Object>）211–213
  Collections.sort() 534, 539

HashMap（HashMap）533
HashSet（HashSet）533
LinkedHashMap（LinkedHashMap）533
LinkedList（LinkedList）533
List（List）557
Map（Map）557, 567
parameterized types（参数化类型）137
Set（Set）557
TreeSet（TreeSet）533

Collections.sort() 534, 539
Comparator 551
compare() 553

Comparable 547, 566
and TreeSet 566
compareTo() method（compareTo()方法）549

Comparator 551, 566
and TreeSet 566

compare() 553

compareTo() 549

comparing with ==（用==号来比较）86

compiler（编译器）2
about（介绍）18
java -d（java -d）590

concatenate（连接）17

concrete classes（创建类）`200–210

conditional expressions（条件表达式）10, 11, 13

constants（常量）282

constructors（构造函数）
about（介绍）240
chaining（链接）250–256
overloaded（重载）256
superclass（父类）250–256

contracts（合约）190–191, 218

cups（杯子）51

curly braces（curly braces）10

D

DailyAdviceClient（DailyAdviceClient程序代码）480

DailyAdviceServer（DailyAdviceServer程序代码））484

dancing girl（跳舞女孩）316

dates（日期）
Calendar（Calendar）303
methods（方法）305
formatting（格式）301
GregorianCalendar（GregorianCalendar）303
java.util.Date 303

deadlock（死锁）516

deadly diamond of death（致命方块）223

declarations（声明）
about（介绍）50
exceptions（异常）335–336
instance variables（实例变量）50

default access（默认存取）668

default value（默认值）84

deployment options（部署的选择）582, 608

deserialized objects 441（见序列化）

directory structures（目录结构）
packages（包）589
servlets（servlets）626

doctor（医生）169

dot operator（圆点运算符）
reference（引用）54

double（double）51

duck（鸭子）277
construct（构造）242
garbage collect（垃圾收集器）261

ducking exceptions（ducking异常）335

E

EJB 631

encapsulation（封装）
about（介绍）79–82
benefits（优点）80

end of book（结束）648

enumerations（枚举）671–672
enums（枚举）671–672
equality（相等）560
  and hashCode()（hashCode()方法）561
equals()（equals()方法）561
equals()（equals()方法）
  about（介绍）209
  Object class（Object类）209
event handling（事件处理）357–361
  event object（事件对象）361
  listener interface（监听接口）358–361
  using inner classes（使用内部类）379
event source（事件源）359–361
exceptions（异常）
  about（介绍）320, 325, 338
  catch（catch块）321, 338
  catching multiple exceptions（捕获多重异常）329, 332
  checked vs. runtime（检查和运行期）324
  declaring（声明）335–336
  ducking（ducking掉）335–336
  finally（finally块）327
  flow control（流程控制）326
  handle or declare law（处理或声明）337
  propagating（传播）335–336
  remote exceptions（远程异常）616
  throwing（throw块）323–326
  try（try块）321, 338
executable JAR（可执行的JAR）585–586, 586
  with packages（包）592, 592–593
exercises（练习）
  be the...（是……）88, 118, 266, 310, 395
  code magnets（排排看）20, 43, 64, 119, 312, 349, 467, 524–525
  honeypot 267
  true or false（是非题）311, 348, 466, 602
  what's the declaration（Java声明呢？）231
  what's the picture（图呢？）230
  which layout manager?（哪一个layout管理器）424
  who am I（我是谁？）45, 89, 394
Extreme Programming（极限编程）101

F

File（文件）452
FileInputStream（文件输入流）441（见输入/输出）
FileOutputStream（文件输出流）432
FileReader 454（见输入/输出）
files
  File class（File类）452
  reading from（读取）441, 454
  source file structure（源文件结构）7
  writing to（写入）432, 447
FileWriter 447
File class（File类）452
final（final标识）
  class（类）189, 283
  methods（方法）189, 283
  static variables（静态变量）282
  variables（变量）282, 283
finally block（finally块）327
fireside chats
  about（介绍）18
five minute mystery（见*puzzles*）
float（float类型）51
FlowLayout（FlowLayout布局）403, 408–410
flow control（流程控制）
  exceptions（异常）326
font（字体）406
formatting（格式化）
  dates（日期）301–302
  format specifiers（格式化声明）295–296
  argument（参数）300
  numbers（数字）294–295
  printf()（printf()方法）294
  String.format()（String.format()）294
for loops（for循环）105
fully qualified name（完整名称）154, 157
  packages（包）587

G

garbage collection（垃圾收集器）
about（介绍）40
eligible objects（合适对象）260–263
heap（堆）57, 58
nulling references（空引用）58
reassigning references（再分配引用）58
generics（泛型）540, 542, 568–574
methods（方法）544
wildcards（万用字符）574
getters and setters（getters和setters）79
ghost town（荒源小镇）109
giraffe（长颈鹿）50
girl dreaming（女孩的梦想）
inner classes（内部类）375
Java Web Start（JWS）596
girl in a tub（澡盆中的女孩）177
girl who isn't getting it 182–188
graphics（图形）364–366（见GUI）
Graphics2D class（Graphics2D类）366
Graphics object（Graphics对象）364
GregorianCalendar（GregorianCalendar）303
guessing game（猜猜看游戏）38
GUI（图形用户接口）406
about（介绍）354, 400
animation（动画效果）382–385
BorderLayout（BorderLayout管理器）370–371, 401, 407
BoxLayout（BoxLayout管理器）403, 411
buttons（按钮）405
components（元件）354, 363–368, 400
event handling（事件处理）357–361, 379
FlowLayout（FlowLayout管理器）403, 408
frames（框架）400
graphics（图形）363–367
ImageIcon class（ImageIcon类）365
JButton（JButton组件）400
JLabel（JLabel组件）400
JPanel（JPanel组件）400, 401
JTextArea（JTextArea）414
JTextField（JTextField）413
layout managers（布局管理器）401–412
listener interface（监听接口）358–361
scrolling (JScrollPane)（滚动条）414
Swing（Swing）354
GUI Constants（GUI常量）
ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER 415
ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS 415
GUI methods（GUI方法）
drawImage() 365
fillOval() 365
fillRect() 364
gradientPaint(). *See also* GUI
paintComponent() 364
setColor() 364
setFont() 406
GUI Widgets（GUI部件）354
JButton 354, 405
JCheckBox 416
JFrame 354, 400, 401
JList 417
JPanel 400, 401
JScrollPane 414, 417
JTextArea 414
JTextField 413

H

HAS-A 177–181
hashCode() 561
HashMap 533, 558
HashSet 533, 558
Hashtable 558
heap（堆）
about（介绍）40, 57, 236–238
garbage collection（垃圾收集器）40, 57, 58

I

I/O（输入/输出）

BufferedReader 454, 478
BufferedWriter 453
buffers（缓冲区）453
deserialization（解序列化）441
FileInputStream 441
FileOutputStream 432
FileWriter 447
InputStreamReader 478
ObjectInputStream 441
ObjectOutputStream 432, 437
serialization（序列化）432, 434–439, 437, 446, 460
streams（流）433, 437
with sockets（socket）478

if -else 13

if statement（if语句）13

immutability, Strings
immutability（不变性）661

implements（实现）224

imports
static imports（静态imports）307

import statement（import语句）155, 157

increment（递增）105

inheritance（继承）
about（介绍）31, 166–192
and abstract classes（抽象类）201
animals（动物）170–175
IS-A（IS-A）214, 251
super（super的使用）228

initializing（初始化）
instance variables（实例变量）84
primitives（primitive主数据类型）84
static variables（静态变量）281

inner classes（内部类）
about（介绍）376–386
events（事件）379

inner class threesome（内部类）381

InputStreamReader 478

instance variables（实例变量）
about（介绍）34, 73
declaring（声明）84
default values（默认值）84
initializing（初始化）84
life and scope（生命周期和范围）258–263
local variables vs.（局部变量）236–238, 239
static vs.（静态）277

instantiation（实例化，见对象）

int（int类型）50
primitive（primitive主数据类型）51

Integer. See wrapper

interfaces（接口）
about（介绍）219–227
for serialization（序列化）437
implementing（实现）224, 437
implementing multiple（实现多个接口）226
java.io.Serializable 437

IP address（IP地址，见网络章节）

IS-A 177–181, 251

J

J2EE 631

JAR files
basic commands（基本命令）593
executable（可执行的）585–586, 592
manifest 585
running executable（运行期可执行）586, 592
tool（工具）593
with Java Web Start（JWS）598

Java, about（Java介绍）5, 6

javac（见编译器）

Java in a Nutshell 158–159

java sound 317, 340

Java Web Start（JWS）597–601
jnlp file（jnlp文件）598, 599

Jini 632–635

JNLP 598
jnlp file（jnlp文件）599

JPEG 365

JVM（Java虚拟机）
about（介绍）2, 18

JWS（JWS，见Java Web Start）

K

keywords（关键字）53

L

l 264

layout managers（布局管理器）401–412
- BorderLayout 370–371, 403, 407
- BoxLayout 403, 411
- FlowLayout 403, 408–410

lingerie, exceptions（异常）329

LinkedHashMap 533, 558

LinkedHashSet 558

LinkedList 533, 558

linked invocations（链接的调用）664

List 557

listeners（监听）
- listener interface（监听接口） 358–361

literals, assigning values（赋值）
- primitive（primitive主数据类型）52

local（局部）
- variables（变量）85, 236, 236–238, 258–263

locks（锁）
- object（对象）509
- threads（线程）509

long 51

loops（循环）
- about（介绍）10
- break（break语句）105
- for（for语句）105
- while（while语句）115

lost update problem（丢失的更新问题，见线程）

M

main() 9, 38

make it stick 53, 87, 157, 179, 227, 278

manifest file（manifest文件）585

Map 557, 567

Math class（Math类）
- methods（方法）274–278, 286
- random() 111

memory
- garbage collection（垃圾收集器）260–263

metacognitive tip（学习密诀）33, 108, 325

methods（方法）
- about（介绍）34, 78
- abstract（抽象）203
- arguments（参数）74, 76, 78
- final 283
- generic arguments（泛型参数）544
- on the stack（栈）237
- overloading（重载）191
- overriding（覆盖）32, 167–192
- return（返回值）75, 78
- static（静态）274–278

midi 317, 340–346, 387–390

midi sequencer 340–346

MINI Cooper 504

modifiers（修饰符）
- class（类）200
- method（方法）203

multidimensional arrays（多维数组）670

multiple inheritance（多重继承）223

multiple threads（见线程）

music（见midi）

mystery（见puzzles）

N

naming（命名，见RMI）
- classes and interfaces（类和接口）154–155, 157
- collisions（冲突）587
- packages（包）587

networking（网络）
- about（介绍）473
- ports（端口）475

sockets（socket）475

new 55

null

reference（引用）262

numbers

formatting（格式化）294–295

O

ObjectOutputStream 432, 437

objects（对象）

about（介绍）55

arrays（数组）59, 60, 83

comparing（比较）209

creation（创建）55, 240–256

eligible for garbage collection（可垃圾回收的）260–263

equality（相等）560

equals() 209, 561

life（生命周期）258–263

locks（锁）509

Object class（Object类）

about（介绍）208–216

equals() 561

hashCode() 561

overriding methods（覆盖方法）563

object graph（对象图）436, 438

object references对象引用）54, 56

assignment（赋值）55, 262

casting（转换）216

comparing（比较）86

equality（相等）560

nulling 262

polymorphism（多态）185–186

OO（面向对象）

contracts（合约）190–191, 218

deadly diamond of death（致命方块）223

design（设计）34, 41, 79, 166–191

HAS-A 177–181

inheritance（继承）166–192

interfaces（接口）219–227

IS-A 177–181, 251

overload（重载）191

override（覆盖）167–192

polymorphism（多态）183, 183–191, 206–217

superclass（父类）251–256

operators（运算符）

and autoboxing 291

bitwise（位）660

comparison（比较）151

conditional（条件）11

decrement（递减）115

increment（递增）105, 115

logical（逻辑）151

shift（移位）660

overload（重载）191

constructors（构造函数）256

override（覆盖）

about（介绍）32, 167–192

polymorphism（多态，见多态）

P

packages（包）154–155, 157, 587–593

directory structure（目录结构）589

organizing code（代码组织）589

paintComponent() 364–368

parameter（参数，见arguments）

parameterized types（参数化类型）137

parsing an int（见wrapper）

parsing text with String.split() 458

pass-by-copy（见pass-by-value）

pass-by-value 77

phrase-o-matic 16

polymorphism（多态）183–191

abstract classe（抽象类）206–217

and exceptions（异常）330

arguments and return types（参数和返回类型）187

references of type Object（Object引用类型）211–213

pool puzzle（见puzzles）

ports（端口）475

prep code（伪码）99–102

primitives（primitive主数据类型）53
- == operator（==运算符）86
- autoboxing 288–289
- boolean 51
- byte 51
- char 51
- double 51
- float 51
- int 51
- ranges（域）51
- short 51
- type 51

primitive casting（转换primitive主数据类型）
- explicit primitive 117

printf() 294

PrintWriter 479

private
- access modifier（存取修饰符）81

protected 668

public
- access modifier（存取修饰符）81, 668

puzzles
- five minute mystery（5分钟短剧）92, 527, 674
- Java cross 22, 120, 162, 350, 426, 603
- pool puzzle（泳池迷宫）24, 44, 65, 91, 194, 232, 396

Q

quiz card builder 448, 448–451

R

rabbit 50

random() 111

ready-bake code（现成码）112, 152–153, 520

reference variables（引用变量，见对象引用转换）216

registry, RMI 615, 617, 620

remote control（远程控制）54, 57

remote interface（远程接口，见RMI）

reserved words（保留字）53

return types（返回类型）
- about（介绍）75
- polymorphic（多态）187
- values（值）78

risky code（风险码）319–336

RMI
- about（介绍）614–622
- client（客户端）620, 622
- compiler（编译器）618
- Jini（见Jini）
- Naming.lookup() 620
- Naming.rebind(). *See also* RMI
- registry 615, 617, 620
- remote exceptions 616
- remote implementation 615, 617
- remote inteface 615, 616
- rmic 618
- skeleton 618
- stub 618
- UnicastRemoteObject 617
- universal service browser（通用服务浏览器）636–648

rmic（见RMI）

run()
- overriding in Runnable interface（覆盖Runnable接口）494

Runnable interface（Runnable接口）492
- about（介绍）493
- run() 493, 494
- threads（线程）493

runnable thread state（运行中的线程状态）495

Ryan and Monica（杰纶与沛晨）505–506
- introduction（介绍）505–506

S

scary objects（怪物对象）200

scheduling threads（线程调度）
- scheduling（调度）496–498

scope
variables（变量）236–238, 258–263
scrolling (JScrollPane)（滚动）414
serialization（序列化）434–439, 446
deserialization（解序列化）460
interface（接口）437
ObjectInputStream（见输入/输出）
objectOutputStream 432
objects（对象）460
object graph（对象图）436
reading（见输入/输出）
restoring 460（见输入/输出）
saving 432
serialVersionUID 461
transient 439
versioning 460, 461
writing 432
server（服务器）
socket 483. *See also* socket
servlet 625–627
Set 557
importance of equals()（equals()的重要性）561
importance of hashCode()（hashCode()的重要性）561
short 51
short circuit logical operators 151
sink a dot com 96–112, 139–150
skeleton. *See* RMI
sleep() 501–503
sleeping threads（sleep线程）501–503
snowboard 214
socket
about（介绍）475
addresses（地址）475
creating（创建）478
I/O（输入/输出）478
ports（端口）475
reading from（读数据）478
server（服务器）483
TCP/IP 475
writing to（写数据）479
sorting（排序）
Collections.sort() 534, 539, 547
Comparable interface（Comparable接口）547, 549
Comparator 551, 553
TreeSet 564–566
source files（源文件）
structure of（结构）7
specifiers（说明）
format specifiers（格式化说明）295, 298
argument specifier（参数说明）300
stack（堆栈）
heap vs.（堆）236
methods on（方法）237
scope（范围）236
threads（线程）490
trace 323
static
enumerated types（枚举类型）671
initializer（初始化）282
Math class methods（Math类的方法）274–278
methods（方法）274–278
static imports（307
variables（变量）282
streams（流）433.（见输入/输出）
String
arrays（数组）17
concatenating（连接）17
methods（方法）669
parsing（解析）458
String.format() 294–297
String.split() 458
StringBuffer/StringBuilder
methods（方法）669
stub（见RMI）
subclass（子类）
about（介绍）31, 166–192
super（父类）228
about（介绍）31
superclass（父类）

about（介绍）166–192, 214–217, 228
super constructor（父类构造函数）250–256
Swing. *See* GUI
synchronized
methods（方法）510（见线程）
syntax（语法）
about（介绍）10, 12
System.out.print() 13
System.out.println() 13

T

talking head（关于head的谈论）203
TCP ports（TCP端口）475
Telluride 30
testing
extreme programming（极限编程）101
text
parsing with String.split()（用String.split()解析）458
read from a file（读取文件，见输入/输出）
write to a file（写入文件）447
text area (JTextArea) 414
text field (JTextField) 413
Thread.sleep() 501–503
threads（线程）
about（介绍）489–515
deadlock（死锁）516
locks（锁）509
lost update problem（丢失更新问题）512–514
run() 493, 494
Runnable 492, 493, 494
Ryan and Monica problem（杰伦和沛晨的问题）505–507
scheduling（调度）496, 496–498
sleep() 501–503
stack（堆栈）490–491
start() 492
starting 492
states（状态）495, 496
summary（概要）500, 517
synchronized（同步化）510–512
unpredictability（不同预测性）498–499
throw
exceptions（异常）323–326
throws 323–326
transient 439
TreeMap 558
TreeSet 533, 558, 564–566, 566
try
blocks（块）321, 326
type（类型）50
parameter（参数）137, 542, 544
type-safety（类型安全）540
and generics（泛化）540

U

universal service browser（通用服务浏览器）636–648

V

variables（变量）
assigning（赋值）52, 262
declaring（声明）50, 54, 84, 236–238
local（局部）85, 236–238
nulling 262
primitive（primitive主数据类型）51, 52
references（引用）54, 55, 56, 185–186
scope（范围）236–238
static（见static）
variable declarations（声明变量）50
instance（实例）84
primitive（primitive主数据类型）51
reference（引用）54
virtual method invocation（虚拟方法调用）175

W

web start. *See* Java Web Start

W

while loops（while循环）11, 115

wildcard（万用字符）574

wine（酒）202

wrapper（包装）287

- autoboxing 288–289
- conversion utilities 292
- Integer.parseInt() 104, 106, 117

writing（见输入/输出）

这不是结束

带着你的大脑来参观

wickedlysmart.com

